JOURNAL OF COMPUTER AND SYSTEM SCIENCES 50, 165–187 (1995)

Implication Problems for Functional Constraints on Databases Supporting Complex Objects*

MINORU ITO

Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630-01, Japan

AND

GRANT E. WEDDELL

Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Received October 22, 1991; revised December 1, 1993

Virtually all semantic or object-oriented data models assume that objects have an identity separate from any of their parts and allow users to define complex object types in which part values may be any other objects. In G. E. Weddell (*ACM Trans. Database* Systems **17**, No. 1 (1992), 32–64), a more general form of functional dependency is proposed for such models in which component attributes may correspond to descriptions of property paths, called *path functional dependencies* (PFDs). The main contribution of the reference is a sound and complete axiomatization for PFDs when databases may be infinite. However, a number of issues were left open which are resolved in this paper. We first prove that the same axiomatization remains complete if PFDs are permitted empty left-hand sides, but that this is *not* true if logical consequence is defined with respect to finite databases. We then prove that the implication problem for arbitrary PFDs is decidable. The proof suggests a means of characterizing an important function closure which is then used to derive an effective procedure for constructing a deterministic finite state automaton representing the closure. The procedure is further refined to efficient polynomial time algorithms for the implication problem for cases in which antecedent PFDs are a form of complex key constraint.

1. INTRODUCTION

There are at least two problems with the relational model when used for more involved applications [11]: users must introduce properties of objects to serve as their means of reference; and all relationships between objects must be expressed indirectly in terms of these properties. Virtually all semantic or object-oriented data models overcome these problems by assuming that objects have an identity separate from any of their parts and by allowing users to define complex object types in which part values may be any other objects [1, 2, 12, 15, 17]. A more general language of functional constraints for a data model supporting the definition of such complex object types was considered in [20]. One feature of the model in common with a number of others [4, 5, 9] is that a database is viewed as a labeled directed graph. The idea is that objects and property values correspond to vertices and arcs, respectively. The constraint language is novel since it allows descriptions of property value paths in a database graph to occur as component attributes. Members of the languages are therefore referred to as *path functional dependencies* (PFDs).

An example of a collection of complex object types which can be defined in terms of the data model in [20] is illustrated by the UNIVERSITY *schema graph* in Fig. 1, which characterizes information about student course enrollment at a hypothetical university. Informally, each complex object type is represented by a labeled vertex, together with a number of outgoing labeled arcs. The vertex label is a *class* name and each outgoing arc represents a function which is total on the "form" class and single-valued on the "to" class.

Some examples of PFDs over the UNIVERSITY schema are listed in Table I. The initial four entries use the special property value path descriptor, `Id`, to assert "keys." For example, the first is satisfies by a database graph only if no two departments have the same name. (Similar constraints might also be given for students and professors.) The fifth and sixth PFDs are conseqences of a requirement that professors only teach courses offered by their own departments, while the last is justified by physical reality—it asserts that a student cannot be enrolled in two separate courses at the

* This research was supported in part by a grant from the Ministry of Education, Science and Culture of Japan, by the Natural Sciences and Engineering Research Council of Canada, and by Bell-Northern Research Ltd.

0022-0000/95 $6.00

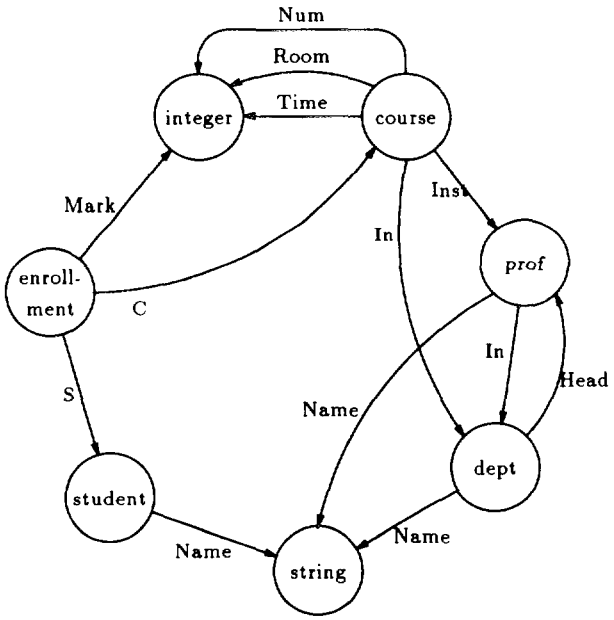

FIG. 1. The UNIVERISTY schema graph.

same time. Note that the last may also be viewed as a form of complex or embedded key constraint. This becomes more apparent if one considers an alternative wording for the constraint: "in the context of the enrollments for a particular student, no two courses are given at the same time."

There are many reasons why it is important to be able to reason about functional dependecies beyond their use in relational schema design and evaluation. An early application in query optimization involves determining *minimal covers* of selection and join conditions [3]. Several authors have also suggested how they may be used to aid in automatically inserting "cut" operators in access plans based on nested iteration [10, 13, 14, 20], in detecting search conditions for complex object indices [20], and in deducing when "project" operations (or DISTINCT modifiers) can be eliminated from a query experession [20].

An example of the last case, from [7], will help to motivate some of the results in this paper. To begin, consider the following query on the UNIVERSITY database.

Find all students enrolled in some course taught at the same time as some other course numbered 101 that is taught by a professor in the CS department.

TABLE I

PFDs over the UNIVERSITY Schema

```
dept(Name → Id)
dept(Head → Id)
course(Num In → Id)
enrollment(S C → Id)
course(Inst.In → In)
course(In → Inst.In)
enrollment(S C.Time → C)
```

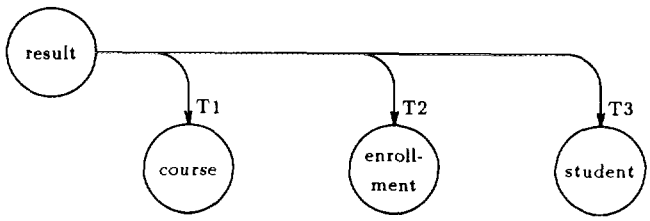

FIG. 2. A query on the UNIVERISTY schema.

An access plan for the request, expressed in terms of a complex object algebra [8, 9, 16, 18], might be given as

$$T1 := \sigma_{\text{Ins.In.Name} = \text{'CS'} \wedge \text{Num} = 101}\, \texttt{course}$$

$$T2 := T1 \bowtie_{\text{Time} = \text{C.Time}} \texttt{enrollment}$$

$$T3 := \pi_{\{S\}}\, T2$$

The problem is to determine if it is possible for the number of tuples in T2 to ever exceed the number of tuples in T3. To see how PFD theory can help solve the problem, consider an abstraction of the query as the additional result object type on the UNIVERSITY schema illustrated by Fig. 2. In addition, include in the query abstraction the following list of four PFDs:

```
result(T3 → T2.S)
result(T1.Time → T2.C.Time)
result(∅ → T1.Inst.In.Name T1.Num)
result(T1 T2 T3 → Id)
```

Each is mandated in turn by the projection, join, and selection operators, and on the grounds that any *particular* combination of T1, T2, and T3 tuples need only be recorded at most once by a result object. The issue is clearly resolved if the key PFD

```
result(T3 → Id)
```

is a logical consequence of these and the other PFDs listed in Table I. In fact, the set of inference axioms proposed in [20] is sufficient to determine that this is indeed the case.

The main contribution of this earlier work is a proof that the inference axioms are complete. However, the proof of completeness depended on two assumptions: that the left-hand sides of antecedent PFDs are non-empty; and that databases can be of infinite size. In Section 3, we prove a positive and a negative result concerning these assumptions. The positive result is that allowing PFDs with empty left-hand sides does not alter the theory. (The example above demonstrates at least one use of such PFDs in abstracting selection conditions in queries.) The negative result is that

the inference axioms are *not* complete if logical consequence is defined with respect to finite databases only; that is, we prove that the implication problem and finite implication problem for PFDs are not equivalent.

Our main result relates to another issue which was left open in [20]. In Section 4, we prove that the implication problem for arbitrary PFDs is decidable, which we believe to be important new evidence that PFDs are a feasible concept in complex object databases. The proof suggests a means of characterizing an important function closure. In Section 5, we derive an effective procedure for constructing a deterministic finite state automaton representing the closure. The procedure is further refined in Section 6, in which we derive polynomial time algorithms for the implication problem for cases in which antecedent PFDs are key constraints. Our summary comments are given in Section 7.

2. DEFINITIONS AND BASIC CONCEPTS

We begin by presenting the syntax of our data model, commonly referred to as the *data definition language* (DDL). An instance of the DDL defines a space of possible databases. In our case, an element of this space will correspond to a labeled directed graph.

DEFINITION 1 (*syntax—the* DDL). A *class schema* S consists of a finite set of complex object types of the form

$$C\{P_1: C_1, ..., P_n: C_n\}$$

in which C is a class name, and the set $\{P_1, ..., P_n\}$ are its properties, written Props(C). Each property P_i is unique in a given class, and its type, written Type(C, P_i), is the name C_i of another (not necessarily distinct) class. The set of names of classes in S is denoted Classes(S). By convention, only the first letter of property names will be capitalized.

The declarations for a UNIVERSITY class schema outlined pictorially in Fig. 1 are formally defined in Table II. Note how several properties, such as S or C, have non-built-in classes their range, and how the In and Head properties demonstrate that problem schema may be cyclic.

TABLE II

The UNIVERSITY Schema

```
enrollment{S:student, C:course, Mark:int}
student{Name:string}
course{In:dept, Inst:prof, Room:int, Num:int,
   Time:int}
dept{Name:string, Head:prof}
prof{In:dept, Name:string}
string{ }
int{ }
```

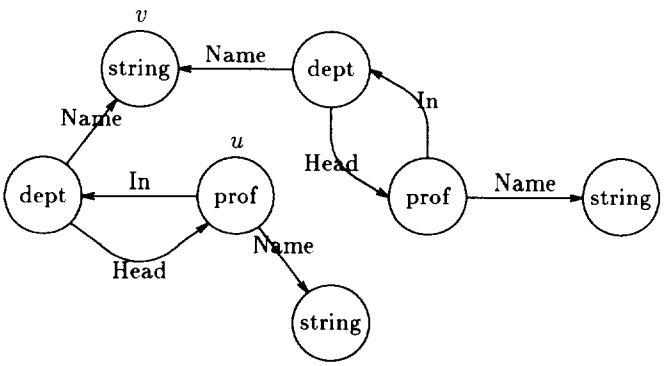

FIG. 3. A database for the UNIVERSITY schema.

DEFINITION 2 (*semantics—a database*). A database for class schema S is a (possibly infinite) directed graph $G(V, A)$ with vertex and edge labels corresponding to class and property names, respectively. G must also satisfy the following three constraints, where the class name label of a vertex v is denoted $l_{Cl}(v)$:

1. (Property value integrity). If $u \xrightarrow{P} v \in A$, then $P \in \text{Props}(l_{Cl}(u))$ and $l_{Cl}(v) = \text{Type}(l_{Cl}(u), P)$.
2. (Property functionality). If $u \xrightarrow{P} v$, $u \xrightarrow{P} w \in A$, then $v = w$.
3. (Property value completeness). If $u \in V$, then there is an arc $u \xrightarrow{P} v \in A$ for every $P \in \text{Props}(l_{Cl}(u))$.

The UNIVERSITY schema graph in Fig. 1 is one possible database for the UNIVERSITY schema. In this case, a single object exists for each complex object type. The directed graph of Fig. 3 depicts another possibility in which two departments have the same name. (Note that different string vertices represent different strings, although the particular strings involved, or integers for that matter, are never important to our presentation.)

DEFINITION 3. A *path function pf* over class schema S is either Id (short for *identity*), or a finite sequence of property names separated by dots. (We assume Id does not correspond to the name of any property in S. The identity path function is our means of referring to property value paths of zero length.) Their composition and length are defined as follows:

$$pf_1 \circ pf_2 = \begin{cases} pf_1 & \text{if } pf_2 \text{ is Id,} \\ pf_2 & \text{if } pf_1 \text{ is Id,} \\ pf_1 \cdot pf_2 & \text{otherwise.} \end{cases}$$

$$\text{len}(pf) = \begin{cases} 0 & \text{if } pf \text{ is Id,} \\ 1 + \text{len}(pf_1) & \text{otherwise, where } pf = pf_1 \circ P, \\ & \text{for some property } P. \end{cases}$$

Let X be a set of path functions $\{pf_1, ..., pf_2\}$. We write $pf \circ X$ to denote $\{pf \circ pf_1, ..., pf \circ pf_n\}$.

Note that the composition operator is clearly associative; that is, $pf_1 \circ (pf_2 \circ pf_3) = (pf_1 \circ pf_2) \circ pf_3$. For example, with the UNIVERSITY schema, `S∘Name` is the path function `S.Name`, and both `Id∘C` and `C∘Id` are the path function `C`. The expression `Id∘C∘Room` denotes either `(Id∘C)∘Room` or `Id∘(C∘Room)`, and in both cases is the path function `C.Room`. The following identity on *len* is also a straightforward consequence of our definitions:

$$\text{len}(pf_1 \circ pf_2) = \text{len}(pf_1) + \text{len}(pf_2).$$

DEFINITION 4. A path $u \rightarrow \cdots \rightarrow w \xrightarrow{P} v$ in a database $G(V, A)$ for class schema **S** is *described* by a path function pf iff either (1) the path consists of a single vertex u and pf is `Id`, or (2) pf is $pf_1 \circ P$, where $u \rightarrow \cdots \rightarrow w$ is described by pf_1.

For example, `In.Name` and `In.Head.In.Name` are path functions which describe paths from vertex u to v in Fig. 3. Now consider that `Name.In` is also a path function according to our definitions, but that no path can exist in any database for the UNIVERSITY schema which is described by `Name.In`. In [20], a subset of path functions for a given class schema **S**, denoted PF(**S**) below, is defined and proven to satisfy a completeness property for databases over **S**: any path in any database for **S** can be described by a path function in PF(**S**), and any path function in PF(**S**) describes a path in some database for **S**. The same reference also proves an important sense in which the composition operator remains closed over PF(**S**). Both the results are reproduced as Lemma 1 and Lemma 2 below.

DEFINITION 5. The set of *well-formed path functions* PF(**S**) over class schema **S** is the smallest set of path functions over **S** satisfying the following two conditions:

1. $\texttt{Id} \in \text{PF}(\mathbf{S})$, where
   (a) $\text{Dom}(\texttt{Id}) \stackrel{\text{def}}{=} \text{Classes}(\mathbf{S})$, and
   (b) $\text{Ran}(C, \texttt{Id}) \stackrel{\text{def}}{=} C$, for all $C \in \text{Classes}(\mathbf{S})$.
2. If $pf \in \text{PF}(\mathbf{S})$, $C \in \text{Dom}(pf)$, and $P \in \text{Props}(\text{Ran}(C, pf))$, then $pf \circ P \in \text{PF}(\mathbf{S})$, where
   (a) $\text{Dom}(pf \circ P) \stackrel{\text{def}}{=} \{C_1 \in \text{Dom}(pf) \mid P \in \text{Props}(\text{Ran}(C_1, pf))\}$, and
   (b) $\text{Ran}(C_1, pf \circ P) \stackrel{\text{def}}{=} \text{Type}(\text{Ran}(C_1, pf), P)$, for all $C_1 \in \text{Dom}(pf \circ P)$.

Capital letters X, Y, and Z are used to denote finite subsets of PF(**S**) for some class schema **S**, and XY, for example, denotes the union of path functions mentioned in X and Y. By a slight abuse of notation, we write PathFuncs(C) to denote all path functions $pf \in \text{PF}(\mathbf{S})$ where $C \in \text{Dom}(pf)$, for $C \in \text{Classes}(\mathbf{S})$. A class schema **S** is *cyclic* iff there exists $pf \in \text{PF}(\mathbf{S}) - \{\texttt{Id}\}$ and $C \in \text{Dom}(pf)$, where $C = \text{Ran}(C, pf)$. (A simple consequence is that **S** is cyclic iff PF(**S**) is infinite.)

Note that the subset of well-formed path functions for cyclic class schema, however, continues to be infinite. For example, the UNIVERISTY schema has a well-formed "head of the department" function `In.Head`, a "head of the department of the head of the department" function `In.Head.In.Head`, and so on. Other well-formed UNIVERSITY path functions include

`S, S.Name, C, C.Room, C.Time,`

`C.Inst, C.Inst.In`

and

`C.Inst.In.Head.`

Note that each of these path functions is also in PathFuncs(`enrollment`). Also, for example

$$\text{Dom}(\texttt{Name}) = \{\texttt{prof}, \texttt{dept}, \texttt{student}\}$$

and

$$\text{Ran}(\texttt{enrollment}, \texttt{C.Inst}) = \texttt{prof}.$$

LEMMA 1. (Expressiveness of well-formed path functions —from [20]). *Let $G(V, A)$ be a database for a given class schema* **S**. *If a path $u \rightarrow \cdots \rightarrow v$ exists in G, then there exists a unique $pf' \in \text{PathFuncs}(l_{\text{Cl}}(u))$ describing $u \rightarrow \cdots \rightarrow v$. Also, for every $u \in V$ and $pf'' \in \text{PathFuncs}(l_{\text{Cl}}(u))$, there exists a path $u \rightarrow \cdots \rightarrow v$ in G described by pf''.*

Note that Lemma 1 also asserts that no two distinct paths with common end vertices can be described by the *same* path function (which motivates the use of the phrase "path function," as opposed to, say, "path description"). For example, vertex v in Fig. 3 is the unique vertex reachable from vertex u by a path described by `In.Name`. By a slight abuse of notation, we write u.`In.Name` to denote v, and in general $u.pf$ to denote the unique vertex reachable from u by a path described by pf, whenever $pf \in \text{PathFuncs}(l_{\text{Cl}}(u))$.

LEMMA 2. (Closure of composition—also from [20]). *Assume that $C \in \text{Classes}(\mathbf{S})$, for some class schema* **S**. *Then $pf_1 \in \text{PathFuncs}(C)$, $pf_2 \in \text{PF}(\mathbf{S})$ and $\text{Ran}(C, pf_1) \in \text{Dom}(pf_2)$ iff $pf_1 \circ pf_2 \in \text{PathFuncs}(C)$.*

The remaining definitions in this section present the syntax of our functional constraint language and define satisfaction and logical consequence as they relate to the above graph-based view of databases.

DEFINITION 6. The syntax of a *path functional dependency* (PFD) over class schema **S** is given by $C(X \rightarrow Y)$. Such a constraint is *well-formed* if $XY \subseteq \text{PathFuncs}(C)$. (This definition differs slightly from the one given in [20] —we now admit PFDs with no path functions occuring before the arrow; that is, $X = \varnothing$.)

A *key path functional dependency* (key PFD) satisfies the additional condition that, for any $pf \in Y$, there exists a path function pf' such that $pf \circ pf' = pf''$ for some $pf'' \in X$; that is, that every right-hand side path function is a "prefix" of some left-hand side path function. We say that a key PFD is *simple* if the single path function `Id` occurs on the right-hand side; that is, $Y = \texttt{Id}$. (This definition also differs from the one given in [20]. The notion of a key PFD has been somewhat generalized to include what we have called complex or embedded keys in our introductory comments.)

The PFD $C(X \rightarrow Y)$ is *satisfied* by a database $G(V, A)$ for S iff for any pair of vertices $u, v \in V$, where $l_{\mathrm{Cl}}(u) = l_{\mathrm{Cl}}(v) = C$, whenever $u \cdot pf = v \cdot pf$ for every $pf \in X$, then $u \cdot pf' = v \cdot pf'$ for every $pf' \in Y$. Note that if $X = \varnothing$, then the antecedent is trivially satisfied; in this case, $u \cdot pf' = v \cdot pf'$ for every $pf' \in Y$ unconditionally. For example, PFD of the form $C(\varnothing \rightarrow \texttt{Id})$ is only satisfied by a database with at most one C object. If, on the other hand, $Y = \varnothing$, then the consequent is trivially satisfied; in this case, the PFD is satisfied by *any* database for S.

By Lemma 1, any PFD that is not well formed is always (trivially) satisfied. Also note that a schema graph, such as the one depicted in Fig. 1 for the UNIVERSITY schema, must satisfy *any* well-formed PFD when viewed as a database since no class has more than one object. In contrast, the UNIVERSITY database in Fig. 3 illustrate a violation of the first key PFD in Table I on class `dept` (two distinct department objects have the same name).

DEFINITION 7 (*Logical consequence*). Let F denote a finite set of PFDs over class schema S, and let f denote an arbitrary PFD also over S. Then f is a *logical consequence* of F, written $F \models f$, iff any database for S satisfying all constraints in F must also satisfy f. We say that f is *trivial* if $\varnothing \models f$; that is, if any database for S always satisfies f.

3. ON PROOF THEORIES FOR PFD CONSTRAINTS

3.1. *A Complete Axiomatization for the Implication Problem*

In [20], it was proven that the inference axioms for PFDs listed in Table III are sound and that axioms A1 to A5 are complete if there are no PFDs of the form $C(\varnothing \rightarrow X)$. In this subsection, we extend this earlier work to show that allowing PFDs with empty left-hand sides does not alter the theory; that is, that axioms A1 to A5 in Table III remain complete. A proof theory based on the axioms is given as follows.

DEFINITION 8. Let $F \cup \{C(X \rightarrow Y)\}$ denote a finite set of PFDs over class schema S. There is a derivation of $C(X \rightarrow Y)$ from F, written $F \vdash C(X \rightarrow Y)$, iff $C(X \rightarrow Y)$ is a member of F, or can be derived from F using the inference axioms in Table III. Also, if $X \subseteq \mathrm{PathFuncs}(C)$, for some class C, then X^+ denotes the smallest set containing all $pf \in \mathrm{PathFuncs}(C)$, where $F \vdash C(X \rightarrow pf)$. (Note that X^+ may not be finite.)

TABLE III

Axioms for PFDs

	Name	Definition
A1	(reflexivity)	$\dfrac{Y \subseteq X \subseteq \mathrm{PathFuncs}(C)}{C(X \rightarrow Y)}$
A2	(augmentation)	$\dfrac{C(X \rightarrow Y),\ Z \subseteq \mathrm{PathFuncs}(C)}{C(XZ \rightarrow YZ)}$
A3	(transitivity)	$\dfrac{C(X \rightarrow Y),\ C(Y \rightarrow Z)}{C(X \rightarrow Z)}$
A4	(simple attribution)	$\dfrac{P \in \mathrm{Props}(C)}{C(\texttt{Id} \rightarrow P)}$
A5	(simple substitution)	$\dfrac{P \in \mathrm{Props}(C),\ \mathrm{Type}(C, P)(X \rightarrow Y)}{C(P \circ X \rightarrow P \circ Y)}$
A6	(additivity)	$\dfrac{C(X \rightarrow Y),\ C(X \rightarrow Z)}{C(X \rightarrow YZ)}$
A7	(projectivity)	$\dfrac{C(X \rightarrow YZ)}{C(X \rightarrow Y)}$
A8	(attribution)	$\dfrac{pf \in \mathrm{PathFuncs}(C)}{C(\texttt{Id} \rightarrow pf)}$
A9	(substitution)	$\dfrac{pf \in \mathrm{PathFuncs}(C),\ \mathrm{Ran}(C, pf)(X \rightarrow Y)}{C(pf \circ X \rightarrow pf \circ Y)}$

Both the earlier proof of completeness in [20] and our modification to the proof require the construction and manipulation of a special kind of graph called a *C-tree*.

DEFINITION 9. Let C denote an arbitrary class in Classes(S), for class schema S. A *C-tree* is a (possibly infinite) directed graph $G_C(V_C, A_C)$ for S constructed as follows.

Step 1. For each $pf \in \mathrm{PathFuncs}(C)$, add vertex u with $l_{\mathrm{Cl}}(u)$ assigned $\mathrm{Ran}(C, pf)$, and with an additional vertex label $l_{\mathrm{Pf}}(u)$ (called its *path function labeling*) assigned pf. The single vertex v with $l_{\mathrm{Pf}}(v) = \texttt{Id}$ is denoted as $\mathrm{Root}(G_C)$.

Step 2. For each $u, v \in V_C$, where $l_{\mathrm{Pf}}(u) = pf$ and $l_{\mathrm{pf}}(v) = pf \circ P$, add $u \xrightarrow{P} v$ to A_C.

Note that, given a class schema S, a C-tree for S is unique up to isomorphism.

A *partial C-tree* is a subtree of a C-tree with the same root. (A partial C-tree may be a C-tree as a special case.) For any vertex u in a partial C-tree, we refer to $\mathrm{len}(l_{\mathrm{Pf}}(u))$ as the *depth* of u.

An example partial `course`-tree for the UNIVERSITY schema appears in Fig. 4. Note that we have indicated the additional path function labeling for each vertex in parenthesis below the class labeling. Also note that, although this tree is finite, a full `course`-tree would necessarily be infinite since PathFuncs(`course`) is infinite.

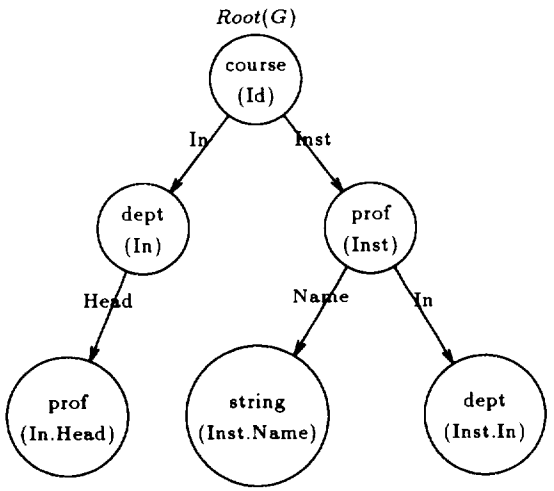

FIG. 4. A partial course-tree for the UNIVERSITY schema.

The properties satisfied by a partical C-tree that are important to our presentation are given in the following lemma.

LEMMA 3. *Let $G(V, A)$ be a partial C-tree for class schema* S, *where* $C \in$ Classes(S). *Then the following three conditions hold*:

CT1. *For every $u \in V$ and every $pf \in$ PathFuncs($l_{\mathrm{Cl}}(u)$), if $u \cdot pf \in V$, then $l_{\mathrm{Pf}}(u) \circ pf = l_{\mathrm{Pf}}(u \cdot pf)$.*

CT2. *For every $u \in V$, $u =$ Root$(G) \cdot l_{\mathrm{Pf}}(u)$.*

CT3. *If G is a* full *C-tree, then G is a database for* S *such that, for every $pf \in$ PathFuncs(C), there is a unique vertex $u \in V$ such that $u =$ Root$(G) \cdot pf$.*

Proof. (See proof of Lemma 7 in [20].) ∎

A simple consequence of condition CT2 is that the depth of any vertex in G is its path length from Root(G). For example, the depth of the single `string` vertex in the partial `course`-tree of Fig. 4 is len(`Inst.Name`) ($=2$).

Now let F be a finite set of PFDs over S which contains PFDs with empty left-hand sides. Along the same line as proving Theorem 2 in [20], it can be shown that inference axioms A1 to A5 are sound; that is, $F \vdash C(X \rightarrow Y)$ implies $F \models C(X \rightarrow Y)$ for any PFD $C(X \rightarrow Y)$ over S. Hence inference axioms A6 to A9 are also sound by Lemma 5 in [20].

In general, to prove that inference axioms A1 to A5 are complete, it suffices to show that $F \nvdash C(X \rightarrow Y)$ implies $F \not\models C(X \rightarrow Y)$; that is, if $Y \not\subseteq X^+$, then to construct a database for S that satisfies F but not $C(X \rightarrow Y)$. We may also assume, without loss of generality, that no PFD in F is trivial, and furthermore, by additivity A6 and projectivity A7, that the right-hand side of every PFD in F consists of a single path function; that is, that every PFD in F is of the form $C(Z \rightarrow pf)$.[1] The earlier completeness proof in [20] constructed such a database, called a *two-C-tree*, from two copies of a (full) C-tree. The important conditions satisfied by a two-C-tree database G are

1. G contains two vertices $R1$ and $R2$ in which $l_{\mathrm{Cl}}(R1) = l_{\mathrm{Cl}}(R2) = C$ and such that $R1 \cdot pf = R2 \cdot pf$ iff $pf \in X^+$ for every $pf \in$ PathFuncs(C). (Thus, if $Y \not\subseteq X^+$, then G must fail to satisfy $C(X \rightarrow Y)$.)

2. G satisfies F (provided that no PFD in F has an empty left-hand side).

The main difficulty with a two-C-tree $G(V, A)$, if F has PFD contraints with empty left-hand sides, is that G might contain distinct vertices $u, v \in V$ in which $l_{\mathrm{Cl}}(u) = l_{\mathrm{Cl}}(v) = C'$, for some $C' \in$ Classes(S), and for which $u = R1 \cdot pf_1 = R2 \cdot pf_1$ and $v = R1 \cdot pf_2 = R2 \cdot pf_2$, for some $pf_1, pf_2 \in$ PathFuncs(C) (i.e., $pf_1, pf_2 \in X^+$). Then, for example, G will fail to satisfy F should it contain the constraint $C'(\varnothing \rightarrow$ `Id`$)$.[2]

Roughly, our refinement to the earlier proof overcomes this problem by modifying the definition of a two-C-tree. The modification, called a *two-C-graph*, will satisfy the condition that, for each $C' \in$ Classes(S), there is a *unique* vertex v such that $R1 \cdot pf = R2 \cdot pf = v$ for every $pf \in$ PathFuncs(C) with $pf \in X^+$ and Ran(C, pf) $= C'$. We prove that a two-C-graph database will satisfy all PFDs in F, including any with empty left-hand sides.

As in the earlier case of a two-C-tree, the construction of a two-C-graph starts with two copies of a (full) C-tree. In addition, another special kind of database which we referred to in the introduction as a *schema graph* is also used.

DEFINITION 10. A *schema graph* for S is a directed graph $G_{\mathrm{S}}(V_{\mathrm{S}}, A_{\mathrm{S}})$ constructed as follows:

Step 1. For each $C \in$ Classes(S), add vertex u with $l_{\mathrm{Cl}}(u)$ assigned C.

Step 2. For each $u, v \in V_{\mathrm{S}}$, where $l_{\mathrm{Cl}}(u) = C$ and $l_{\mathrm{Cl}}(v) =$ Type(C, P), add $u \xrightarrow{P} v$ to A_S.

Note that given a class schema S, a schema graph for S is unique up to isomorphism.

LEMMA 4. *Let $G_{\mathrm{S}}(V_{\mathrm{S}}, A_{\mathrm{S}})$ be a schema graph for class schema* S. *Then the following holds*:

SG1. *For each $C \in$ Classes(S), there is a unique vertex $v \in V_{\mathrm{S}}$ such that $l_{\mathrm{Cl}}(v) = C$.*

SG2. *G_{S} is a database for* S.

Proof. Obvious. ∎

[1] No PFD in F has an empty *right*-hand side, since, by reflexivity A1, such a PFD would be trivial.

[2] The problem generalizes to any PFD constraint of the form $C'(\varnothing \rightarrow pf)$.

a{ B:b, C:c }
b{ A:a, D:d }
c{ G:b, E:e }
d{ F:f }
e{ }
f{ }

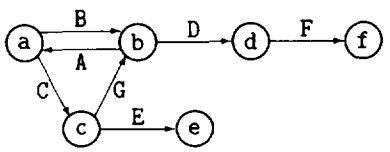

FIG. 5. A class schema and its schema graph.

For example, the database illustrated in Fig. 1 is a schema graph for the UNIVERSITY class schema listed in Table II. Another example of a class schema and corresponding schema graph appears in Fig. 5.

The definition of a two-C-graph relies on the following "suffix closure" condition for X^+.

LEMMA 5. *For* $pf \in \text{PathFuncs}(C)$, *if* $pf \in X^+$, *then* $pf \circ pf' \in X^+$ *for every* $pf \circ pf' \in \text{PathFuncs}(C)$.

Proof. Let $C' = \text{Ran}(C, pf)$. Then $pf \circ pf' \in \text{PathFuncs}(C)$ implies $pf' \in \text{PathFuncs}(C')$, by Lemma 2. By attribution A8, we can derive $C'(\texttt{Id} \rightarrow pf')$, and thus $C(pf \rightarrow pf \circ pf')$ by substitution A9. Since $pf \in X^+$, $F \vdash C(X \rightarrow pf)$. By transitivity A3, $C(X \rightarrow pf)$ and $C(pf \rightarrow pf \circ pf')$ imply $C(X \rightarrow pf \circ pf')$. Hence $pf \circ pf' \in X^+$. ∎

Now consider where there is a PFD $C(X \rightarrow Y)$ such that $F \not\vdash C(X \rightarrow Y)$; that is $Y \not\subseteq X^+$. Then, since $pf = \texttt{Id} \circ pf$ for any $pf \in \text{PathFuncs}(C)$, it follows from Lemma 5 that

$$\texttt{Id} \notin X^+. \tag{3.1}$$

DEFINITION 11. Let $F \cup \{C(X \rightarrow Y)\}$ denote a set of PFDs such that $F \not\vdash C(X \rightarrow Y)$. A *two-C-graph* is a (possibly infinite) directed graph $G(V, A)$ constructed as follows:

Step 1. Construct two C-trees $G_C^1(V_C^1, A_C^1)$ and $G_C^2(V_C^2, A_C^2)$ and a schema graph $G_S(V_S, A_S)$. $\text{Root}(G_C^1)$ and $\text{Root}(G_C^2)$ are denoted by $R1$ and $R2$, respectively.

Step 2. Let $V_i = \{v \in V_C^i \mid l_{Pf}(v) \notin X^+\}$ for $i = 1, 2$. Note that $R1 \in V_1$ and $R2 \in V_2$ by (3.1). Let $A_i = \{u \xrightarrow{P} v \in A_C^i \mid u, v \in V_i\}$ for $i = 1, 2$. Add all vertices in $V_S \cup V_1 \cup V_2$ to V and all arcs in $A_S \cup A_1 \cup A_2$ to A.

Step 3. For each $u \in V_1 \cup V_2$ and each $P \in \text{Props}(l_{Cl}(u))$, where $u \xrightarrow{P} v \notin A$ for any $v \in V$, add an arc $u \xrightarrow{P} w$ to A, where $w \in V_S$ and $l_{Cl}(w) = \text{Type}(l_{Cl}(\text{u}), P)$. Note that w is unique by condition SG1 of Lemma 4.

It should be clear from this definition that a two-C-graph is symmetric with respect to $R1$ and $R2$. An outline of the form of a two-C-graph is illustrated in Fig. 6, in which we denote a (possibly infinite) partial C-tree consisting of V_i and A_i by the label $G_i(V_i, A_i)$, where $i = 1, 2$. The arcs added in Step 3 are also indicated.

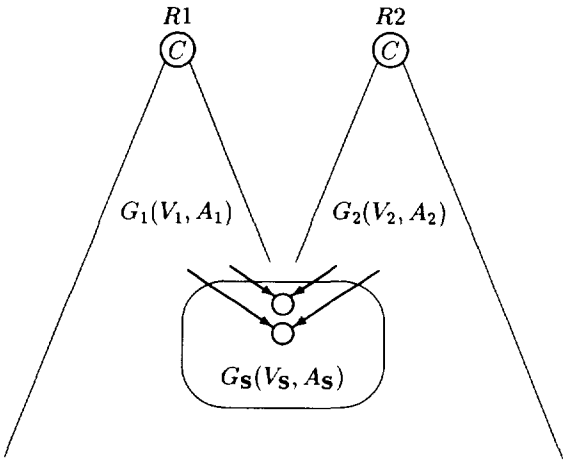

FIG. 6. General form of a two-C-graph.

LEMMA 6. *The two-C-graph* $G(V, A)$ *is a database for* S *satisfying the following three conditions*:

TCG1. (a) *If* $u \in V_S$, *then* $u \cdot pf \in V_S$ *for every* $pf \in \text{PathFuncs}(l_{Cl}(u))$.

(b) *If* $u \in V_i$, *then exactly one of* $u \cdot pf \in V_i$ *and* $u \cdot pf \in V_S$ *holds for every* $pf \in \text{PathFuncs}(l_{Cl}(u))$, *where* $i = 1$ *or* 2.

TCG2. *For every* $pf \in \text{PathFuncs}(C)$, $pf \in X^+$ *iff* $R1 \cdot pf \in V_S$ *iff* $R1 \cdot pf = R2 \cdot pf$.

TCG3. *For any pair of distinct vertices* $u, v \in V$, *where* $l_{Cl}(u) = l_{Cl}(v)$, *if* $u \cdot pf = v \cdot pf$ *for some* $pf \in \text{PathFuncs}(l_{Cl}(u))$, *then* $u \cdot pf \in V_S$.

Proof. Since G_S is a database for S, it can be proven with the same line of argument used in the proof of Lemma 8 in [20] that G is also a database for S.

Now consider TCG1. Clearly, just after Step 2 in Definition 11, for every $u \xrightarrow{P} v \in A$, exactly one of $\{u, v\} \subseteq V_1$, $\{u, v\} \subseteq V_2$, and $\{u, v\} \subseteq V_S$ holds. Furthermore, for each arc $u \xrightarrow{P} v$ added to A in Step 3, $u \in V_1 \cup V_2$ and $v \in V_S$. Thus, after Step 3, for every $u \xrightarrow{P} v \in A$, exactly one of $v \in V_S$, $\{u, v\} \subseteq V_1$, and $\{u, v\} \subseteq V_2$ must be true. TCG1(a) therefore follows. Since $V_i \cap V_S = \varnothing$, TCG1(b) also follows.

Consider TCG2. We first prove that $pf \in X^+$ iff $R1 \cdot pf \in V_S$. Since $R1 \in V_1$, it follows from TCG1(b) that $R1 \cdot pf \in V_S$ iff $R1 \cdot pf \notin V_1$. Thus, it suffices to show that

$$pf \notin X^+ \quad \text{iff} \quad R1 \cdot pf \in V_1. \tag{3.2}$$

It follows from definition of V_1 that, for every $v \in V_C^1$, $l_{Pf}(v) \notin X^+$ iff $v \in V_1$. This implies (3.2) since there is a one-to-one correspondence between V_C^1 and $\text{PathFuncs}(C)$ according to conditions CT2 and CT3 of Lemma 3. We next

prove that $R1 \cdot pf \in V_S$ iff $R1 \cdot pf = R2 \cdot pf$. Assume that $R1 \cdot pf \in V_S$. Since the two-C-graph is symmetric with respect to $R1$ and $R2$, $R1 \cdot pf \in V_S$ implies both $R2 \cdot pf \in V_S$ and $l_{Cl}(R1 \cdot pf) = l_{Cl}(R2 \cdot pf)$. Thus $R1 \cdot pf = R2 \cdot pf$ by condition SG1 of Lemma 4. Conversely, since $R1 \in V_1$, $R2 \in V_2$, and $V_1 \cap V_2 = \varnothing$, it follows from TCG1(b) that $R1 \cdot pf = R2 \cdot pf$ implies $R1 \cdot pf \in V_S$.

Finally consider TCG3. Assume that $u \cdot pf = v \cdot pf$ but that $u \cdot pf \notin V_S$ for two distinct vertices $u, v \in V$. By TCG1(a), $u \cdot pf \notin V_S$ implies $u \notin V_S$, that is, $u \in V_1 \cup V_2$. Assume without loss of generality that $u \in V_1$. By TCG1(b), $u \in V_1$ and $u \cdot pf \notin V_S$ imply $u \cdot pf$ $(= v \cdot pf) \in V_1$. By TCG1(a) and (b), $v \cdot pf \in V_1$ implies $v \in V_1$; that is, the three vertices u, v, and $u \cdot pf$ $(= v \cdot pf)$ are in V_1. Since $G_1(V_1, A_1)$ is a partial C-tree, it follows from condition CT2 of Lemma 3 that

$$w = R1 \cdot l_{Pf}(w) \qquad \text{for every} \quad w \in V_1. \tag{3.3}$$

Furthermore by condition CT1 of that lemma, $u \cdot pf = v \cdot pf$ implies that $l_{Pf}(u) \circ pf = l_{Pf}(v) \circ pf$, that is, $l_{Pf}(u) = l_{Pf}(v)$, and, therefore, that $u = v$, a contradiction with our assumption above that u and v are distinct vertices. Hence, if $u \neq v$ and $u \cdot pf = v \cdot pf$, then $u \cdot pf \in V_S$. ∎

LEMMA 7. *For $pf \in \text{PathFuncs}(C)$, if there is a* PFD *$C'(Z \rightarrow pf') \in F$ such that $C' = \text{Ran}(C, pf)$ and $pf \circ Z \subseteq X^+$, then $pf \circ pf' \in X^+$.*

Proof. Since $C' = \text{Ran}(C, pf)$, $C'(Z \rightarrow pf')$ implies $C(pf \circ Z \rightarrow pf \circ pf')$ by substitution A9. Since $pf \circ Z \subseteq X^+$, $F \vdash C(X \rightarrow pf \circ Z)$ by definition. Hence, $C(X \rightarrow pf \circ Z)$ and $C(pf \circ Z \rightarrow pf \circ pf')$ imply $C(X \rightarrow pf \circ pf')$ by transitivity A3. That is, $pf \circ pf' \in X^+$. ∎

THEOREM 1. *Inference axioms A1 to A5 are sound and complete, even in the case that there are PFDs with empty left-hand sides.*

Proof. It suffices to prove that a two-C-graph $G(V, A)$ is a database satisfying F but not $C(X \rightarrow Y)$.

We first show that G does not satisfy $C(X \rightarrow Y)$. Since $X \subseteq X^+$ by reflexivity $A1$, it follows from condition TCG2 of Lemma 6 that $R1 \cdot pf = R2 \cdot pf$ for every $pf \in X$. Conversely, since $Y \not\subseteq X^+$, it follows from condition TCG2 that $R1 \cdot pf \neq R2 \cdot pf$ for some $pf \in Y$. Hence G does not satisfy $C(X \rightarrow Y)$.

We next show that G satisfies F. Assume that G does not satisfy a PFD $C'(Z \rightarrow pf) \in F$. Then there are two distinct vertices $u, v \in V$ such that

$$l_{Cl}(u) = l_{Cl}(v) = C', \tag{3.4}$$

$$u \cdot pf_Z = v \cdot pf_Z \qquad \text{for every} \quad pf_Z \in Z,^3 \tag{3.5}$$

$$u \cdot pf \neq v \cdot pf. \tag{3.6}$$

[3] If $Z = \varnothing$, then this condition holds trivially.

Since $u \cdot pf \notin V_S$ or $v \cdot pf \notin V_S$ by (3.4), (3.6), and condition SG1 of Lemma 4, assume without loss of generality that

$$u \cdot pf \in V_1. \tag{3.7}$$

Then $u \in V_1$ by conditions TCG1(a) and (b). Thus $u = R1 \cdot l_{Pf}(u)$ by (3.3) in the proof of Lemma 6. Furthermore, it follows from property functionality and property value completeness for the database G that, for every $pf' \in \text{PathFuncs}(l_{Cl}(u))$, there is a unique vertex $w \in V$ such that $w = u \cdot pf'$. Hence,

$$u \cdot pf' = R1 \cdot l_{Pf}(u) \circ pf' \qquad \text{for every} \quad pf' \in \text{PathFuncs}(l_{Cl}(u)). \tag{3.8}$$

In particular, $u \cdot pf = R1 \cdot l_{Pf}(u) \circ pf$. Then, by (3.7) and condition TCG2,

$$l_{Pf}(u) \circ pf \notin X^+. \tag{3.9}$$

Since (1) $\text{Ran}(C, l_{Pf}(u)) = l_{Cl}(u) = C'$ by (3.4) and (2) $C'(Z \rightarrow p) \in F$, it follows from Lemma 7 that (3.9) implies $l_{Pf}(u) \circ Z \not\subseteq X^+$; that is, there exists $pf_Z \in Z$ such that $l_{Pf}(u) \circ pf_Z \notin X^+$. Then $R1 \cdot l_{Pf}(u) \circ pf_Z \notin V_S$ by condition TCG2; that is, $u \cdot pf_Z \notin V_S$ by (3.8). Conversely, since (1) $u \neq v$ by assumption and (2) $l_{Cl}(u) = l_{Cl}(v)$ by (3.4), it follows from condition TCG3 that (3.5) implies $u \cdot pf_Z \in V_S$, a contradiction. Therefore, G satisfies F. ∎

3.2. *The Inequivalence of the Finite and the Infinite Implication Problem*

In this subsection, we prove that the inference axioms for PFDs listed in Table III are *not* complete if databases with infinitely many objects are disallowed. In particular, we exhibit a class schema S and finite set $F \cup \{C(X \rightarrow Y)\}$ of PFDs over S such that $F \not\vdash C(X \rightarrow Y)$, but in which $C(X \rightarrow Y)$ is necessarily satisfied by any *finite* database for S that satisfies F.

DEFINITION 12. Let $F \cup \{C(X \rightarrow Y)\}$ denote a finite set of PFDs over class schema S. $C(X \rightarrow Y)$ is a *finite* logical consequence of F, written $F \models_{\text{finite}} C(X \rightarrow Y)$, iff any *finite* database for S satisfying F must also satisfy $C(X \rightarrow Y)$.

LEMMA 8. *Let* S *consist of the following two complex object types*:

```
a{A:a, B:b}
b{ }
```

Then `a(A.B→Id)` $\not\models$ `a(B→Id)` *and* `a(A.B→Id)` $\models_{\text{finite}}$ `a(B→Id)`.

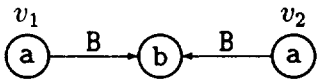

FIG. 7. A subgraph of $G(V, A)$.

Proof. Since a(A.B → Id) is a simple key PFD, the closure of B can be computed efficiently by Theorem 6 in Section 6. In fact, it is easy to verify that $B^+ = \{B\}$. Thus: a(A.B → Id) $\not\models$ a(B → Id).

Assume that a(A.B → Id) $\not\models_{\text{finite}}$ a(B → Id). By definition, there must exist a finite database $G(V, A)$ for **S** that satisfies a(A.B → Id) but not a(B → Id). Then G contains a subgraph given in Fig. 7, where v_1 and v_2 are distinct. There are two general case to be considered.

Case 1. Consider where both v_1 and v_2 have in-arcs labeled "A." This implies that there exist two vertices v_1' and v_2' with "A" arcs to v_1 and v_2, respectively. Since v_1 and v_2 are distinct, v_1' and v_2' are also distinct by property functionality. Figure 8 illustrates one of the possible specific case for G that satisfy these conditions in which v_1' and v_2' are distinct from v_1 and v_2. In the remaining cases, v_i coincides with v_j', where $i = 1$ or 2, and $j = 1$ or 2 (except when this implies $v_1' = v_2'$). However, each of these specific cases implies that G could not satisfy a(A.B → Id), no matter how the other part of G might be constructed, a contradiction.

Case 2. Now consider where either v_1 has no in-arc labeled "A" or v_2 has no in-arc labeled "A." Assume without loss of generality that v_1 has no in-arc labeled "A." By property values completeness, there is an arc $v_1 \xrightarrow{A} u_1 \in A$, where $l_{CI}(u_1) = \text{Type}(l_{CI}(v_1), A) = \text{Type}(a, A) = a = l_{CI}(v_1)$. By a similar argument, there is an arc $u_1 \xrightarrow{A} u_2 \in A$, where $l_{CI}(u_2) = a$. Since G is finite, repeating the above argument will eventually yield a sequence $v_1 \xrightarrow{A} u_1 \xrightarrow{A} \cdots \xrightarrow{A} u_i \xrightarrow{A} \cdots \xrightarrow{A} u_i$ in which u_i occurs twice but all other vertices occur at most once. This implies that the two arcs of the sequence directed to u_i must be distinct. Let $w \xrightarrow{A} u_i$ and $w' \xrightarrow{A} u_i$ denote the arcs. Note that w and w' must be distinct by construction. Figure 9 illustrates one of the possible specific cases for G that satisfy these conditions in which both w and w' are distinct from u_i. (Recall that u_i must have an outgoing arc labeled "B" by property value completeness.) In the remaining cases, either w or w' coincides with u_i. And again, each of these specific cases implies that G could not satisfy a(A.B → Id), regardless of how the other part of G might be constructed, a contradiction.

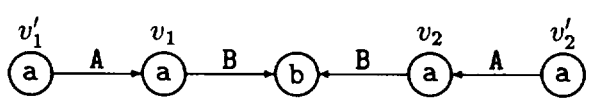

FIG. 8. A possible subgraph of $G(V, A)$ in case 1.

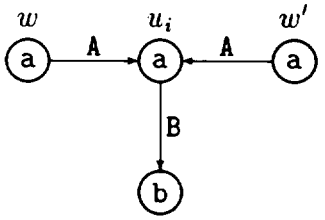

FIG. 9. A possible subgraph of $G(V, A)$ in case 2.

Therefore, there is no finite database that satisfies a(A.B → Id) but does not satisfy a(B → Id). That is, a(A.B → Id) $\models_{\text{finite}}$ a(B → Id). ∎

By Lemma 8, we have the following theorem.

THEOREM 2. *Finite logical implication for PFDs is different from (infinite) logical implication for PFDs, even if all given PFDs are simple key PFDs.*

Theorem 2 implies that inference axioms A1 to A5, although sound, are no longer complete for finite logical implication for PFDs.

4. DECIDABILITY OF THE INFINITE IMPLICATION PROBLEM FOR ARBITRARY PFDS

A semi-decision procedure for the infinite implication problem for arbitrary PFDs is given in [20]. If this problem were equivalent to the finite implication problem, then the existence of this procedure would immediately imply the decidability of both problems [6]. Unfortunately, the results of the previous section show that this is not the case. We shall now resolve the infinite case in this section; we prove that the inifinite implication problem for arbitrary PFDs is decidable.

Our proof is based on a variation of the above-mentioned semi-decision procedure which we will define as function MARK below. The function maps a partial C-tree to a version of partial C-tree, as defined below.

DEFINITION 13. A partial C-tree $G(V, A)$ is *marked* if each $v \in V$ has an additional Boolean valued *mark* label, denoted *Mark*(v). We refer to a vertex v as *marked* (resp. *unmarked*) if *Mark*(v) has the value *true* (resp. *false*).

The resuls of applying MARK relates to a partial order, given in the following, over the set of partial C-trees induced by the marked status of their vertices.

DEFINITION 14. Let $G_1(V_1, A_1)$ and $G_2(V_2, A_2)$ be marked partial C-trees. We write $G_1 \preccurlyeq G_2$ to mean that, for every $pf \in \text{PathFuncs}(C)$, if $\text{Root}(G_1) \cdot pf$ is marked, then $\text{Root}(G_2) \cdot pf$ is marked; that is, the marked vertices in V_1 are *covered* by the marked vertices in V_2. We say G_1 *is*

smaller than G_2 if $G_1 \preccurlyeq G_2$ and $G_2 \not\preccurlyeq G_1$. If $G_1 \preccurlyeq G_2$ and $G_2 \preccurlyeq G_1$, then we write $G_1 \equiv G_2$.

DEFINITION 15. Let $G(V, A)$ be a partial C-tree and let X be a finite subset of PathFuncs(C). Then MARK($G(V, A)$, X) denotes a *smallest* marked version of G (with respect to $\preccurlyeq$) such that every $v \in V$ satisfies the following three conditions:

M1. If $l_{\mathrm{Pf}}(v) \in X$, then v is marked.

M2. If v has an ancestor which is marked, then v is marked.

M3. If there is a PFD $C'(Z \rightarrow pf) \in F$ and a (not necessarily proper) ancestor u of v such that (1) $C' = l_{\mathrm{Cl}}(u)$, (2) $u \cdot pf = v$, and (3) $u \cdot pf_Z$ is marked for every $pf_Z \in Z$,[4] then v is marked.

Each of conditions M1 to M3 can be treated as a *transformation rule* for a vertex v, by making v marked if v is unmarked and satisfies the antecedent of the condition. For convenience, conditions M1, M2, and M3 are called *rules* M1, M2, and M3, respectively, when they are treated as transformation rules. Then MARK(G, X) is computed by the following (not necessarily constructive) procedure.

PROCEDURE GENMARK($G(V, A)$, X).

Input. A partial C-tree $G(V, A)$ and a finite subset X of PathFuncs(C).

Output. MARK(G, X).

Step 1 (*Initialization*). Make all vertices in V unmarked.

Step 2. Apply rules M1 and M3 repeatedly to G, until every vertex in V satisfies conditions M1 to M3.

Note that for each of rules M1 to M3, once a vertex v satisfies the antecedent of the rule, then v will continue to satisfy the same antecedent, regardless of how the rules might be applied to G. This holds since no rules change the status of a vertex from one that is marked to one that is unmarked. In other words, the rules are Church–Rosser. Thus, the final selection of marked vertices in V after a call to GENMARK are uniquely determined and will not depend on the order in which these rules are applied. Furthermore, since every vertex in V is initially unmarked, GENMARK must yield the smallest G (with respect to $\preccurlyeq$) such that every vertex in V satisfies conditions M1 to M3; that is, the procedure returns MARK(G, X). (This implies that MARK(G, X) is uniquely defined.) Conversely, we may view the set of all such marked vertices as determined by a (possibly infinite) sequence of applications of these rules.

For an example of computing function MARK using

[4] If $Z = \emptyset$, then condition (3) holds trivially.

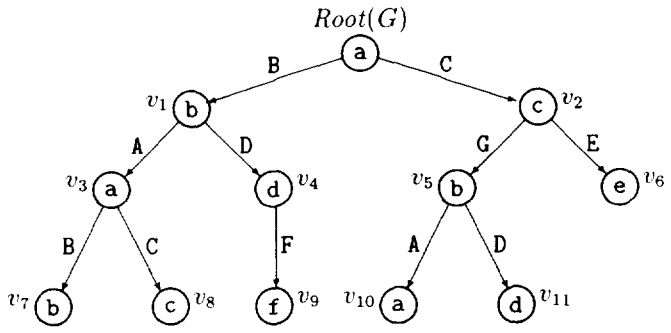

FIG. 10. A partial a-tree $G(V, A)$.

Procedure GENMARK, recall the class schema appearing in Fig. 5, and assume that F consists of the following PFDs:

f_1: a(B.D C.G → Id)

f_2: a(B.D.F C.G.D → C.E)

f_3: b(D.F → A)

f_4: d(F → Id).

Now consider the result of evaluating GENMARK(G, {B.D, C.G.D}), where G is the partial a-tree appearing in Fig. 10. The marked status of each vertex in V computed by this expression is determined as follows: Initially, all vertices in V are unmarked in Step 1. Consider what is executed in Step 2. Since $l_{\mathrm{Pf}}(v_4) = \mathtt{B.D}$ and $l_{\mathrm{Pf}}(v_{11}) = \mathtt{C.G.D}$, applying rule M1 to v_4 and v_{11} makes the vertices marked. After that, we do not need to apply rule M1 to any vertex in V. Now consider what happens by applying the other rules. Since vertex v_9 has an ancestor v_4 which is marked, rule M2 implies that v_9 is marked. Rule M3 now applies for two other vertices. The first relates to vertex Root(G) and PFD f_2 since (1) $l_{\mathrm{Cl}}(\mathrm{Root}(G)) = \mathtt{a}$ and (2) both Root(G).B.D.F ($= v_9$) and Root(G).C.G.D ($= v_{11}$) are marked. The second relates to vertex v_1 and PFD f_3 since (1) $l_{\mathrm{Cl}}(v_1) = \mathtt{b}$ and (2) v_1.D.F ($= v_9$) is marked. Thus, by rule M3, vertices Root(G).C.E ($= v_6$) and v_1.A ($= v_3$) must be marked. Finally, since v_7 and v_8 have the marked ancestor v_3, rule M2 implies that vertices v_7 and v_8 are marked.

At this point, observe that it is unnecessary to change the marked status of any vertices in V in order to satisfy either condition M1, M2, or M3. For example, although vertex v_4 and PFD f_4 satisfy the antecedent for a "firing" of rule M3, since (1) $l_{\mathrm{Cl}}(v_4) = \mathtt{d}$ and (2) v_4.F ($= v_9$) is marked, this firing would have no effect on G since vertex v_4.Id ($= v_4$) is already marked. Thus, the above procedure returns the copy G of the partial a-tree with vertices v_3, v_4, v_6, v_7, v_8, v_9, and v_{11} marked and the remainder unmarked.

The next lemma relates the path function labeling of a

marked vertex in the partial C-tree in MARK(G, X) to its membership in an important closure.

LEMMA 9. *Let* $\tilde{G}(\tilde{V}, \tilde{A})$ *denote* MARK$(G(V, A), X)$, *where* G *is a partial* C*-tree and* X *is a finite subset of* PathFuncs(C). *If a vertex* $v \in \tilde{V}$ *is marked, then* $l_{\mathrm{Pf}}(v) \in X^+$.

Proof. Note that the set of marked vertices in $\tilde{V}$ can be obtained by a sequence of applications of rules M1 to M3 in GENMARK. We prove the lemma by induction on this sequence.

Basis. Since every vertex in V is initially unmarked, the basis case is trivial.

Induction. Consider the nth application of either rule M1, M2, or M3 that results in making a vertex v marked. Since $X \subseteq X^+$ by reflexivity A1, if rule M1 applies, then the antecedent $l_{\mathrm{Pf}}(v) \in X$ implies $l_{\mathrm{Pf}}(v) \in X^+$; thereby the lemma follows. If rule M2 applies, then v has an ancestor u which is marked. Condition CT2 of Lemma 3 implies that $u = \mathrm{Root}(G) \cdot l_{\mathrm{Pf}}(u)$ and $v = \mathrm{Root}(G) \cdot l_{\mathrm{Pf}}(v)$. Since u is an ancestor of v, there is a path function $pf \in \mathrm{PathFuncs}(l_{\mathrm{Cl}}(u))$ such that $u \cdot pf = v$. Thus, by condition CT1 of Lemma 3, $l_{\mathrm{Pf}}(v) = l_{\mathrm{Pf}}(u) \circ pf$. Since u is marked, the induction hypothesis implies $l_{\mathrm{Pf}}(u) \in X^+$. Therefore, by Lemma 5, $l_{\mathrm{Pf}}(u) \circ pf\ (= l_{\mathrm{Pf}}(v))$ is in X^+, and the lemma follows

Now consider where rule M3 applies. Then there is a PFD $C'(Z \rightarrow pf) \in F$ and an ancestor u of v such that (1) $C' = l_{\mathrm{Cl}}(u)$, (2) $u \cdot pf = v$, and (3) $u \cdot pf_Z$ is marked for every $pf_Z \in Z$. Since $u = \mathrm{Root}(G) \cdot l_{\mathrm{Pf}}(u)$ by condition CT2 of Lemma 3, condition (3) implies $l_{\mathrm{Pf}}(u) \circ Z \subseteq X^+$ by the induction hypothesis. It then follows from Lemma 7 and condition (1) that $l_{\mathrm{Pf}}(u) \circ pf \in X^+$. Since $l_{\mathrm{Pf}}(v) = l_{\mathrm{Pf}}(u) \circ pf$ by condition (2) and condition CT1 of Lemma 3, the lemma again follows. ∎

We now prove that the set of path function labels of all marked vertices in the result defined by function MARK coincides with X^+ if the first argument to MARK is a (full) C-tree.

LEMMA 10. *Let* $\tilde{G}_C(\tilde{V}_C, \tilde{A}_C) = \mathrm{MARK}(G_C(V_C, A_C), X)$, *where* G_C *is a* C*-tree and* X *is a finite subset of* PathFuncs(C), *and let* Marked *denote the set*

$$\{l_{\mathrm{Pf}}(v) \mid v \in \tilde{V}_C \text{ and } \mathrm{Mark}(v)\}.$$

Then, Marked $= X^+$.

Proof. Assume that Marked $\neq X^+$. Since Marked $\subseteq X^+$ by Lemma 9, the assumption implies that Marked $\subset X^+$ ($A \subset B$ means that A is a proper subset of B). By Theorem 5 in [20], there must exist at least one PFD $C(Z \rightarrow pf) \in F_1(C) \cup F_2(C)$ such that $Z \subseteq$ Marked and $pf \notin$ Marked. Here, $F_1(C)$ is the set of PFDs of the form "$C(pf' \rightarrow pf' \circ P)$", where pf', $pf' \circ P \in \mathrm{PathFuncs}(C)$, and $F_2(C)$ is the set of PFDs that can be derived from F by a single use of substitution A9. Hence, there are two cases to consider.

Case 1 (where $C(Z \rightarrow pf) \in F_1(C)$). The PFD must have the form "$C(pf' \rightarrow pf' \circ P)$"; that is, $Z = \{pf'\}$ and $pf = pf' \circ P$. Since $Z \subseteq$ Marked by assumption, $\mathrm{Root}(\tilde{G}_C) \cdot pf'$ is marked. Since $\mathrm{Root}(\tilde{G}_C) \cdot pf'$ is an ancestor of $\mathrm{Root}(\tilde{G}_C) \cdot pf' \circ P$, by condition M2, $\mathrm{Root}(\tilde{G}_C) \cdot pf' \circ P$ must be marked. Hence $pf' \circ P \in$ Marked, which is a contradiction to our assumption that $pf \notin$ Marked.

Case 2 (where $C(Z \rightarrow pf) \in F_2(C)$). By definition of $F_2(C)$, there is a path function $pf_1 \in \mathrm{PathFuncs}(C)$ such that $C' = \mathrm{Ran}(C, pf_1)$, $C'(W \rightarrow pf_2) \in F$, $Z = pf_1 \circ W$, and $pf = pf_1 \circ pf_2$. Let $v = \mathrm{Root}(\tilde{G}_C) \cdot pf_1$. Then $l_{\mathrm{Cl}}(v) = C'$, since $l_{\mathrm{Cl}}(v) = \mathrm{Ran}(C, pf_1)$ by definition. Furthermore, it follows from conditions CT1 and CT2 of Lemma 3 that $v \cdot pf_2 = \mathrm{Root}(\tilde{G}_C) \cdot pf_1 \circ pf_2$ and $v \cdot pf_W = \mathrm{Root}(\tilde{G}_C) \cdot pf_1 \circ pf_W$ for every $pf_W \in W$. Since $Z \subseteq$ Marked by assumption, $\mathrm{Root}(\tilde{G}_C) \cdot pf_1 \circ pf_W$ $(= v \cdot pf_W)$ is marked for every $pf_W \in W$. Thus, by condition M3, $v \cdot pf_2$ $(= \mathrm{Root}(\tilde{G}_C) \cdot pf_1 \circ pf_2)$ must be marked in view of vertex v and the PFD. Hence $pf_1 \circ pf_2 \in$ Marked, which is again a contradiction to our assumption that $pf \notin$ Marked. ∎

Note that MARK(G, X) can easily be constructed, provided G is finite. By Lemma 10, it is therefore decidable whether or not $F \models C(X \rightarrow Y)$ whenever the C-tree G_C is finite. However, no such construction is apparent if G_C is infinite. Note that G_C is infinite iff there are two distinct vertices v_1, v_2 on a path from $\mathrm{Root}(G_C)$ such that $l_{\mathrm{Cl}}(v_1) = l_{\mathrm{Cl}}(v_2) = C'$, for some $C' \in$ Classes(S). Since Classes(S) is finite, it is therfore decidable whether or not G_C is infinite.

Consequently, for the remainder of this section, we focus on the case in which G_C is infinite. We prove the decidability of the infinite implication problem for PFDs using the following line of argument:

As above, let $\tilde{G}_C = \mathrm{MARK}(G_C, X)$, where G_C is a C-tree and X is a finite subset of PathFuncs(C). Given an integer c_1, we can find an integer c_2 such that, for any $pf \in \mathrm{PathFuncs}(C)$, if (1) $\mathrm{len}(pf) \leqslant c_1$ and (2) the "size" of a partial C-tree G is c_2, then $\mathrm{Root}(\tilde{G}_C) \cdot pf$ is marked iff $\mathrm{Root}(\tilde{G}) \cdot pf$ is marked, where $\tilde{G} = \mathrm{MARK}(G, X)$. Since we can construct $\tilde{G}$ if G is finite, we can therefore decide if $\mathrm{Root}(\tilde{G}_C) \cdot pf$ is marked without the need to construct the infinite C-tree G_C.

DEFINITION 16. For $C \in$ Classes(S) and an integer l, a partial C-tree of *depth* l is a partial C-tree obtained from a (full) C-tree by removing any vertex u and its incident arcs whenever the depth of u is greater than l. Furthermore, a partial C-tree of depth *at least* l is a partial C-tree that contains a partial C-tree of depth l as a subtree with the same root. In the context of a partial C-tree G of depth at least l, we write $G[l]$ to denote the subtree of G with the same root

whose depth is l. Finally, we write $VCnt(C, l)$ to denote the number of vertices in a partial C-tree of depth l.

For a C-tree G_C and a partial C-tree G of depth at least l, the above implies that $G_C[l] = G[l]$. To futher illustrate, the tree appearing in Fig. 10 is a partial a-tree of depth 3, and thus $VCnt(\mathrm{a}, 3) = 12$. Note that G coincides with $G_a[3]$, for an a-tree G_a.

DEFINITION 17. Let $G(V, A)$ be a partial C-tree, where $C \in \mathrm{Classes}(\mathbf{S})$. A vertex $u \in V$ is *functionally complete* in G if, for every $P \in \mathrm{Props}(l_{\mathrm{Cl}}(u))$, there is an arc $u \xrightarrow{P} v \in A$ for which $l_{\mathrm{Cl}}(v) = \mathrm{Type}(l_{\mathrm{Cl}}(u), P)$. Otherwise, u is *functionally incomplete* in G.

Note that if G is a partial C-tree of depth at least l, then every vertex in V whose depth is less than l is functionally complete in G. For example, with regard to the partial a-tree in Fig. 10, every vertex whose depth is less than 3 is functionally complete. This partial a-tree is of depth 3. As for vertices of depth 3, vertex v_9 is functionally complete, while vertices v_7, v_8, v_{10}, and v_{11} are functionally incomplete. For example, v_7 is functionally incomplete since there is no arc of the form $v_7 \xrightarrow{\mathrm{A}} u$, even though $l_{\mathrm{Cl}}(v_7) = \mathrm{b}$ and $\mathrm{A} \in \mathrm{Props}(\mathrm{b})$.

For the remainder of this section, we will also refer to the following values, as defined in the context of a class schema $\mathbf{S}$, a set of PFDs F, a class $C \in \mathrm{Classes}(\mathbf{S})$ and a finite subset X of $\mathrm{PathFuncs}(C)$:

$$l_1 \stackrel{\mathrm{def}}{=} \max_{pf \in X} \mathrm{len}(pf),$$

$$l_2 \stackrel{\mathrm{def}}{=} \max_{pf \in \{Z \cup \{pf'\} \mid C(Z \to pf') \in F\}} \mathrm{len}(pf),$$

$$L_2 \stackrel{\mathrm{def}}{=} 1 + \sum_{C' \in \mathrm{Classes}(\mathbf{S})} 2^{VCnt(C', l_2)}.$$

Note that L_2 is finite since a partial C'-tree of depth l_2 is finite. Also note that the value $2^{VCnt(C', l_2)}$ counts the number of different possible *true/false* assignments of $\mathrm{Mark}(v_i)$ for the vertices $\{v_1, \ldots, v_n\}$ in a partial C'-tree of depth l_2; that is, the number of different "marking patterns."

We now present a key lemma, whose proof will be given in the rest of this section.

LEMMA 11. *Let $G_C(V_C, A_C)$ denote a C-tree. Then*

$$\mathrm{MARK}(G_C[l'_1 + l_2 + L_2], X)[l'_1] \equiv \mathrm{MARK}(G_C, X)[l'_1],$$

where l'_1 is any integer not less than l_1.

If Lemma 11 holds, then the implication problem will be decidable by the following argument. Choose $\max_{pf \in (X \cup Y)} \mathrm{len}(pf)$ as the integer l'_1 (which implies that $l'_1 \geq l_1$, as required), and let $\tilde{G}$ and $\tilde{G}_C$ denote the results of evaluating the left- and right-hand sides of the above identity, respectively. Then $\mathrm{Root}(\tilde{G}_C) \cdot pf$ is in $\tilde{G}_C[l'_1]$ for every $pf \in Y$. Furthermore, $\mathrm{Root}(\tilde{G}_C) \cdot pf$ is marked iff $pf \in X^+$ by Lemma 10. Hence Lemma 11 implies that $Y \subseteq X^+$ iff $\mathrm{Root}(\tilde{G}) \cdot pf$ is marked for every $pf \in Y$. That is, the implication problem will be decidable.

Let l be an integer such that $l \geq l'_1 + l_2 + L_2$, and let $G(V, A)$ be a partial C-tree of depth at least l but not of depth at least $l+1$. (G must exist since we have assumed that the C-tree G_C is infinite.) Also let $\tilde{G}(\tilde{V}, \tilde{A}) = \mathrm{MARK}(G, X)$. Since $\tilde{G}$ is a partial C-tree of depth at least l but not of depth at least $l+1$, there is a functionally incomplete vertex $v \in \tilde{V}$ whose depth is l. For each i such that $l'_1 + 1 \leq i \leq l'_1 + L_2$, there is an ancestor v' of v whose depth is i. For such an ancestor v', there corresponds a partial C'-tree of depth l_2 as a subtree with root v', where $C' = l_{\mathrm{Cl}}(v')$, since $\tilde{G}$ is a partial C-tree of depth at least l ($\geq l'_1 + l_2 + L_2$) and the depth of v' is between $l'_1 + 1$ and $l'_1 + L_2$. Let $T(v')$ be the subtree with root v'. By choice of L_2, there are at least two distinct ancestors v_1, v_2 of v such that (1) $l_{\mathrm{Cl}}(v_1) = l_{\mathrm{Cl}}(v_2) = C'$ for some $C' \in \mathrm{Classes}(\mathbf{S})$, (2) the depths of v_1 and v_2 are between $l'_1 + 1$ and $l'_1 + L_2$, and (3)

$$T(v_1)[l_2] \equiv T(v_2)[l_2]. \tag{4.1}$$

Assume without loss of generality that v_1 is an ancestor of v_2 (Fig. 11 illustrates the shape of the partial C-tree $\tilde{G}(\tilde{V}, \tilde{A})$ as discussed thus far), and let $G_R(V_R, A_R)$ be the tree obtained from $\tilde{G}(\tilde{V}, \tilde{A})$ by replacing the subtree $T(v_2)$ with the subtree $T(v_1)$. Then we have the following.

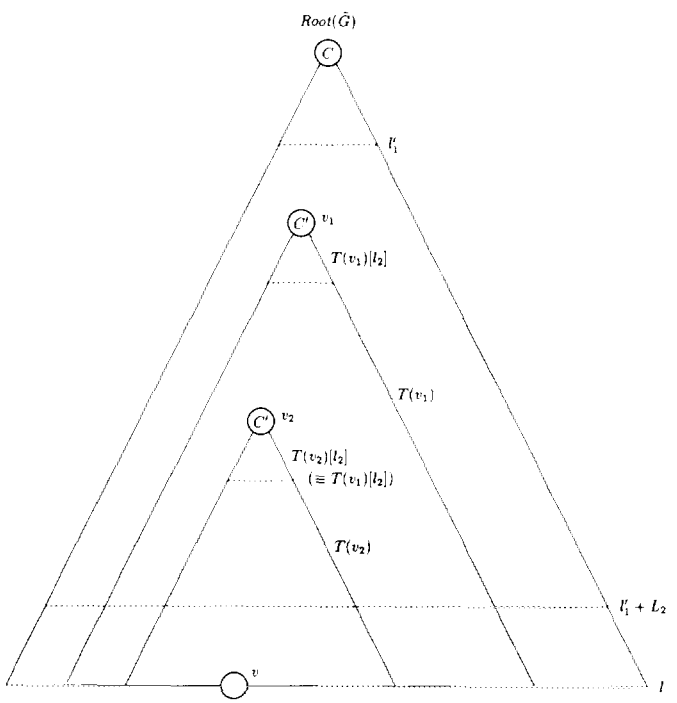

FIG. 11. The C-tree $\tilde{G}(\tilde{V}, \tilde{A})$ in the case of depth l.

LEMMA 12. *Every $u \in V_R$ whose depth is less than l is functionally complete in $G_R(V_R, A_R)$. Also, the number of functionally incomplete vertices of depth l in $G_R(V_R, A_R)$ is less than the number of functionally incomplete vertices of depth l in $\tilde{G}(\tilde{V}, \tilde{A})$.*

Proof. By definition of $\tilde{G}$, for every $u \in \tilde{V}$, if the depth of u is less than l, then u is functionally complete in $\tilde{G}$. Furthermore, v_1 is a proper ancestor of v_2, since v_1 and v_2 are distinct. Thus, by replacing $T(v_2)$ with $T(v_1)$, at least the vertex $\text{Root}(G_R) \cdot l_{\text{Pf}}(v)$ whose depth is l becomes functionally complete in G_R, and, for every $u \in \tilde{V}$ whose depth is less than l, $\text{Root}(G_R) \cdot l_{\text{Pf}}(u)$ remains functionally complete in G_R. The lemma follows. ∎

LEMMA 13. $\text{MARK}(G_R(V_R, A_R), X) \preccurlyeq G_R(V_R, A_R) \preccurlyeq \text{MARK}(G_C(V_C, A_C), X)$. (*Here,* MARK *is applied to a* marked *partial C-tree G_R. By definition, however, the value of* $\text{MARK}(G_R, X)$ *is independent of the marked status of vertices in V_R; that is, if G'_R is an unmarked version of G_R, then* $\text{MARK}(G'_R, X) \equiv \text{MARK}(G_R, X)$.)

Proof. Let $\tilde{G}_R(\tilde{V}_R, \tilde{A}_R)$ and $\tilde{G}_C(\tilde{V}_C, \tilde{A}_C)$ denote $\text{MARK}(G_R, X)$ and $\text{MARK}(G_C, X)$, respectively. For a vertex $u \in \tilde{V}$, the corresponding vertices $\text{Root}(G_R) \cdot l_{\text{Pf}}(u) \in V_R$ and $\text{Root}(\tilde{G}_C) \cdot l_{\text{Pf}}(u) \in \tilde{V}_C$ are denoted by $\langle u \rangle_R$ and $\langle u \rangle_C$, respectively, if the explicit correspondence is necessary. Also, for vertices $u \in V_R$ and $v \in \tilde{V}_C$, let $T_R(u)$ and $T_C(v)$ denote the subtrees of G_R and $\tilde{G}_R$ with roots u and v, respectively. If T is a tree and T' is a subtree of T, then $T - T'$ denotes the tree obtained by removing T' from T. To begin, we claim the following:

Claim 1. (a) $T(v_1) \equiv T_R(\langle v_2 \rangle_R)$.

(b) $\tilde{G}(\tilde{V}, \tilde{A}) - T(v_2) \equiv G_R(V_R, A_R) - T_R(\langle v_2 \rangle_R)$.

Claim 2. (a) Every vertex in $\tilde{G}(\tilde{V}, \tilde{A})$ satisfies conditions M1 to M3.

(b) Every vertex in $T(v_1)$ satisfies conditions M2 and M3.

(c) Every vertex in $\tilde{G}(\tilde{V}, \tilde{A}) - T(v_1)$ satisfies conditions M2 and M3.

Claim 3. Every vertex in $\tilde{G}_C(\tilde{V}_C, \tilde{A}_C)$ satisfies conditions M1 to M3.

Claim 1 follows from construction of G_R. Since $\tilde{G} = \text{MARK}(C, X)$ and $\tilde{G}_C = \text{MARK}(G_C, X)$, Claims 2(a) and 3 follow from definition of MARK. Claim 2(a) implies 2(b) and (c), since $T(v_1)$ and $\tilde{G} - T(v_1)$ are subtrees of $\tilde{G}$, respectively. In the following we prove $\tilde{G}_R(\tilde{V}_R, \tilde{A}_R) \preccurlyeq G_R(V_R, A_R)$ and $G_R(V_R, A_R) \preccurlyeq \tilde{G}_C(\tilde{V}_C, \tilde{A}_C)$, and thus the lemma will hold.

Proof that $\tilde{G}_R(\tilde{V}_R, \tilde{A}_R) \preccurlyeq G_R(V_R, A_R)$. We will prove that every vertex in V_R satisfies conditions M1 to M3. This must imply that $\tilde{G}_R(\tilde{V}_R, \tilde{A}_R) \preccurlyeq G_R(V_R, A_R)$, since $\tilde{G}_R$ $(= \text{MARK}(G_R, X))$ is the smallest version of G_R with respect to $\preccurlyeq$ such that every vertex in $\tilde{V}_R$ satisfies conditions M1 to M3. Assume, otherwise, that there is a vertex in V_R that does not satisfy one of conditions M1 to M3. There are three cases to be considered.

Case 1 (Where condition M1 is not satisfied). Then $\text{Root}(G_R) \cdot pf$ must be unmarked for some $pf \in X$. On the other hand, $\text{Root}(\tilde{G}) \cdot pf$ is marked, since (1) $\text{Root}(\tilde{G}) \cdot pf$ satisfies condition M1 by Claim 2(a), and (2) $pf \in X$. Since (1) the depth of v_2 is greather than l'_1 by definition and (2) $\text{len}(pf) \leqslant l_1 \leqslant l'_1$, the depth of $\langle v_2 \rangle_R$ is greater than the depth of $\text{Root}(G_R) \cdot pf$. Hence, $\text{Root}(G_R) \cdot pf$ is in $G_R(V_R, A_R) - T_R(\langle v_2 \rangle_R)$. Since $\text{Root}(G_R) \cdot pf$ is unmarked, so is $\text{Root}(\tilde{G}) \cdot pf$ by Claim 1(b). However, this contradicts that $\text{Root}(\tilde{G}) \cdot pf$ is marked.

Case 2 (Where condition M2 is not satisfied). Then there are two vertices $\langle u_1 \rangle_R, \langle u_2 \rangle_R \in V_R$ such that (1) $\langle u_1 \rangle_R$ is marked and an ancestor of $\langle u_2 \rangle_R$, and (2) $\langle u_2 \rangle_R$ is unmarked. By Claims 1(a) and 2(b), it is not the case that both $\langle u_1 \rangle_R$ and $\langle u_R \rangle_R$ are in $T_R(\langle v_2 \rangle_R)$. By Claims 1(b) and 2(c), it is not the case that both $\langle u_1 \rangle_R$ and $\langle u_2 \rangle_R$ are in $G_R(V_R, A_R) - T_R(\langle v_2 \rangle_R)$. Furthermore, $\langle u_1 \rangle_R$ is an ancestor of $\langle u_2 \rangle_R$. Thus, the only possibility is that (1) $\langle u_1 \rangle_R$ is marked, an ancestor of $\langle v_2 \rangle_R$, and in $G_R(V_R, A_R) - T_R(\langle v_2 \rangle_R)$, and (2) $\langle u_2 \rangle_R$ is unmarked, a descendant of $\langle v_2 \rangle_R$, and in $T_R(\langle v_2 \rangle_R)$. By the former observation and Claim 1(b), u_1 is therefore marked and in $\tilde{G}(\tilde{V}, \tilde{A}) - T(v_2)$. By the latter observation and Claim 1(a), there is an unmarked vertex w in $T(v_1)$. Then since (1) v_1 is an ancestor of w and (2) w satisfies condition M2 by Claim 2(b), vertex v_1 must be unmarked. On the other hand, since u_1 is marked, by Claim 2(a) and condition M2, all descendants of u_1 should be marked in $\tilde{G}(\tilde{V}, \tilde{A})$. Since $\langle u_1 \rangle_R$ is an ancestor of $\langle v_2 \rangle_R$, u_1 is also an ancestor of v_2; that is, v_2 is a descendant of u_1. Thus v_2 is marked, and v_1 is also marked by (4.1). However, this contradicts that v_1 is unmarked.

Case 3 (Where conditionM3 is not satisfied.) Then, for a vertex $\langle u \rangle_R \in V_R$, there is a PFD $C'(Z \to pf) \in F$ such that $C' = l_{C'}(\langle u \rangle_R)$, $\langle u \rangle_R \cdot pf_Z$ is marked for every $pf_Z \in Z$, and $\langle u \rangle_R \cdot pf$ is unmarked. There are two more specific cases to be considered.

Case 3.1 (Where $\langle u \rangle_R$ is in $T_R(\langle v_2 \rangle_R)$). Since all $\langle u \rangle_R \cdot pf$ and $\langle u \rangle_R \cdot pf_Z$ are descendants of $\langle u \rangle_R$, these vertices are in $T_R(\langle v_2 \rangle_R)$. Hence the unmarked vertex $\langle u \rangle_R \cdot pf$ in $T_R(\langle v_2 \rangle_R)$ must violate condition M3 by virtue of the vertex $\langle u \rangle_R$ and the PFD. However, this contradicts Claims 1(a) and 2(b).

Case 3.2 (Where $\langle u \rangle_R$ is in $G_R(V_R, A_R) - T_R(\langle v_2 \rangle_R)$). We claim that $u \cdot pf$ is unmarked and $u \cdot pf_Z$ is marked for every $pf_Z \in Z$. If $\langle u \rangle_R \cdot pf$ is in $G_R(V_R, A_R) - T_R(\langle v_2 \rangle_R)$, then $u \cdot pf$ is unmarked by Claim 1(b), since $\langle u \rangle_R \cdot pf$ is unmarked. Assume that $\langle u \rangle_R \cdot pf$ is in

$T_R(\langle v_2\rangle_R)$. Since (1) $\langle u\rangle_R$ is in $G_R(V_R, A_R) - T_R(\langle v_2\rangle_R)$ and (2) $\text{len}(pf) \leq l_2$ by choice of l_2, $\langle u\rangle_R \cdot pf$ must be in $T_R(\langle v_2\rangle_R)[l_2]$. Thus, $u \cdot pf$ is in $T(v_2)[l_2]$. Since: (1) $\langle u\rangle_R \cdot pf$ is unmarked, and (2) $T(v_2)[l_2] \equiv T_R(\langle v_2\rangle_R)[l_2]$ by (4.1) and Claim 1(a), $u \cdot pf$ is unmarked. By a similar argument, $u \cdot pf_Z$ is marked for every $pf_Z \in Z$.

By the above argument, in regard to $\tilde{G}(\tilde{V}, \tilde{A})$ and the given PFD, the unmarked vertex $u \cdot pf$ must violate condition M3 by virtue of the vertex u and the PFD, which is in contradiction with Claim 2(a). Our earlier assertion that $\tilde{G}_R(\tilde{V}_R, \tilde{A}_R) \preccurlyeq G_R(V_R, A_R)$ then follows.

Proof that $G_R(V_R, A_R) \preccurlyeq \tilde{G}_C(\tilde{V}_C, \tilde{A}_C)$. It follows from Lemmas 9 and 10 that

$$\tilde{G}(\tilde{V}, \tilde{A}) \preccurlyeq \tilde{G}_C(\tilde{V}_C, \tilde{A}_C). \tag{4.2}$$

This implies $\tilde{G}(\tilde{V}, \tilde{A}) - T(v_2) \preccurlyeq \tilde{G}_C(\tilde{V}_C, \tilde{A}_C) - T_C(\langle v_2\rangle_C)$. Thus, by Claim 1(b), $G_R(V_R, A_R) - T_R(\langle v_2\rangle_R) \preccurlyeq \tilde{G}_C(\tilde{V}_C, \tilde{A}_C) - T_C(\langle v_2\rangle_C)$. What remains to prove is that $T_R(\langle v_2\rangle_R) \preccurlyeq T_C(\langle v_2\rangle_C)$. Since $T(v_2)[l_2] \preccurlyeq T_C(\langle v_2\rangle_C)[l_2]$ by (4.2), it follows from (4.1) that

$$T(v_1)[l_2] \preccurlyeq T_C(\langle v_2\rangle_C)[l_2]. \tag{4.3}$$

From this observation and Claim 1(a), it suffices to show that

$$T(v_1) - T(v_1)[l_2] \preccurlyeq T_C(\langle v_2\rangle_C) - T_C(\langle v_2\rangle_C)[l_2] \tag{4.4}$$

in order to prove $T_R(\langle v_2\rangle_R) \preccurlyeq T_C(\langle v_2\rangle_C)$. It follows from condition CT1 of Lemma 3 that, for a vertex u in $T(v_1)$, there is a path function $pf \in \text{PathFuncs}(l_{C1}(v_1))$ such that $v_1 \cdot pf = u$. To simplify the notation, assume u' denotes the corresponding vertex $\langle v_2\rangle_C \cdot pf$ in $T_C(\langle v_2\rangle_C)$, and let u be a marked vertex in $T(v_1) - T(v_1)[l_2]$. Then (4.4) follows if u' must also be marked, which we prove by induction on the sequence of applications of rules M1 to M3 of the computation of $\tilde{G}$ by Procedure GENMARK.

Basis. Initially, there is no marked vertex in G. Thus (4.4) holds trivially.

Induction. Consider where vertex u must become marked by virtue of the nth application of one of rules M1 to M3. By the induction hypothesis, we may assume that, for $m < n$, if the mth application of a rule marks vertex w in $T(v_1) - T(v_1)[l_2]$, then w' is also marked in $T_C(\langle v_2\rangle_C) - T_C(\langle v_2\rangle_C)[l_2]$. There are three cases to be considered.

Case 1 (Where rule M1 applies). Then u must coincide with $\text{Root}(\tilde{G}) \cdot pf$ for some $pf \in X$. Since (1) the depth of v_1 is greater than l_1', by definition, and (2) $\text{len}(pf) \leq l_1 \leq l_1'$, the depth of v_1 is greater than the depth of $\text{Root}(\tilde{G}) \cdot pf$. Thus, $\text{Root}(\tilde{G}) \cdot pf$ is in $\tilde{G} - T(v_1)$. However, this contradicts our assumption that u is in $T(v_1) - T(v_1)[l_2]$. Thus, this is not the case.

Case 2 (Where rule M2 applies). Then there is an ancestor w of u that has already been marked. There are three subcases to be considered.

Case 2.1 (where w is in $T(v_1)[l_2]$). Then w' is in $T_C(\langle v_2\rangle_C)[l_2]$ and marked by (4.3). Since w' is an ancestor of u', by Claim 3, u' must be marked to satisfy condition M2.

Case 2.2 (where w is in $T(v_1) - T(v_1)[l_2]$). Note that w has already been marked when u is marked, and thus w' is marked by the induction hypothesis. Hence, as in Case 2.1, u' must be marked.

Case 2.3 (where w is in $G(V, A) - T(v_1)$). Since (1) $T(v_1)$ is a tree with root v_1, (2) w is an ancestor of u, and (3) u is in $T(v_1) - T(v_1)[l_2]$, the assumption implies that w is an ancestor of v_1. Thus, condition M2 implies that v_1, as well as u, should both be marked, by virtue of vertex w. Furthermore, since v_1 is in $T(v_1)[l_2]$, $\langle v_2\rangle_C$ is marked by (4.3). Since $\langle v_2\rangle_C$ is an ancestor of u', as in Case 2.1, vertex u' must be marked.

Case 3 (Where rule M3 applies). Then, for an ancestor w of u, there is a PFD $C'(Z \rightarrow pf) \in F$ such that $C' = l_{C1}(w)$, $w \cdot pf = u$, and $w \cdot pf_Z$ is marked for every $pf_Z \in Z$. Since (1) $\text{len}(pf) \leq l_2$ by choice of l_2, and (2) u is in $T(v_1) - T(v_1)[l_2]$ by assumption, the ancestor w of u must be in $T(v_1)$ in order that $w \cdot pf = u$. Thus, each marked vertex $w \cdot pf_Z$ is also in $T(v_1)$. It can be proven along the same line of argument for Cases 2.1 and 2.2 above that each corresponding vertex $w' \cdot pf_Z$ is marked. Hence, by Claim 3, $w' \cdot pf$ must be marked in $\tilde{G}_C(\tilde{V}_C, \tilde{A}_C)$ to satisfy condition M3 by virtue of vertex w' and the PFD. (Note that $u' = w' \cdot pf$.) This completes the induction proof; Lemma 13 now follows. ∎

We are now ready to prove Lemma 11. To begin, consider the following procedure.

PROCEDURE EXTEND(C, X, l_1', N).

Input. A class C, finite subset X of PathFuncs(C) and integers l_1' and N not less than l_1 and $l_1' + l_2 + L_2$, respectively.

Output. A marked partial C-tree $G(V, A)$ of depth at least N.

Step 1. Let $G(V, A) \leftarrow \text{MARK}(G_C(V_C, A_C)[l_1' + l_2 + L_2], X)$, where G_C is a C-tree. (Observe that there is no functionally incomplete vertex in V whose depth is less than $l_1' + l_2 + L_2$.)

Step 2. **for** $l \leftarrow l_1' + l_2 + L_2$ **to** N **do**

Step 3. **while** there is a vertex of depth l that is functionally incomplete in G **do**

begin

Step 3.1. Let v be a vertex of depth l that is functionally incomplete in G. Find two distinct ancestors v_1, v_2 of v satisfying the following three conditions (assuming, without loss of generality, that v_1 is an ancestor of v_2):

1. $l_{CI}(v_1) = l_{CI}(v_2)$,
2. the depths of v_1 and v_2 are between $l_1' + 1$ and $l_1' + L_2$, and
3. $T(v_2)[l_2] \equiv T(v_2)[l_2]$.

Step 3.2. Replace the subtree with root v_2 by the subtree with root v_1.

end

It follows from Lemma 12 that EXTEND must always terminate and yield a marked partial C-tree $G(V, A)$ of depth at least N for a given parameter N. Let $G_N(V_N, A_N)$ denote the result of evaluating EXTEND for a fixed selection C, X, and l_1' of its remaining parameters. Since Lemma 13 applies to each replacement in Step 3.2, we have

$$\begin{aligned}\text{MARK}(G_N(V_N, A_N), X) &\preccurlyeq G_N(V_N, A_N) \\ &\preccurlyeq \text{MARK}(G_C(V_C, A_C), X).\end{aligned}$$

Here, we claim that $\text{MARK}(G_N[N], X) \preccurlyeq \text{MARK}(G_N, X)$. Note that G_N contains $G_N[N]$ as a subtree with the same root. Thus when computing $\text{MARK}(G_N, X)$, we can *simulate* the computation of $\text{MARK}(G_N[N], X)$ by ignoring the vertices in G_N but not in $G_N[N]$, which implies the claim. Hence it holds that

$$\begin{aligned}\text{MARK}(G_N(V_N, A_N)[N], X) &\preccurlyeq G_N(V_N, A_N) \\ &\preccurlyeq \text{MARK}(G_C(V_C, A_C), X).\end{aligned} \tag{4.5}$$

In order to consider what happens as $N \to \infty$, we define the limit of a sequence of partial C-trees. There are two cases to be considered: unmarked and marked versions.

Let $T_1, T_2, \ldots$ be an infinite sequence of finite *unmarked* partial C-trees. Then $\lim_{n\to\infty} T_n$ is defined to be the *smallest* (possibly infinite) unmarked partial C-tree T that contains T_n, for all n, as subtree with the same root. Here, the "smallest" means that if T' is an unmarked partial C-tree that contains all T_n as a subtree with the same root, then T' must also contain T as a subtree with the same root.

Let $T_1, T_2, \ldots$ be an infinite sequence of finite *marked* partial C-trees.Then $\lim_{n\to\infty} T_n$ is defined to be the smallest marked version T of $\lim_{n\to\infty} T_n'$ with respect to $\preccurlyeq$ such that $T_n \preccurlyeq T$ for all n, where T_n' denotes an unmarked version of T_n. That is, if T' is a marked version of $\lim_{n\to\infty} T_n'$ such that $T_n \preccurlyeq T'$ for all n, then $T \preccurlyeq T'$ must also hold.

Let us now consider the infinite sequence $G_N, G_{N+1}, \ldots$ of marked partial C-trees. By definition, $\lim_{n\to\infty} G_n$ exists. Similarly, $\lim_{n\to\infty} \text{MARK}(G_n[n], X)$ exists for the infinite sequence $\text{MARK}(G_N[N], X), \text{MARK}(G_{N+1}[N+1], X), \ldots$ of marked partial C-trees.

Let G_n' be an unmarked version of G_n for $n \geqslant N$. We claim that $\lim_{n\to\infty} G_n'$ coincides with a *full* C-tree. Note that G_n' is a partial C-tree of depth at least n. Thus for every finite subtree T of the full C-tree with the same root, if we choose an integer n which is not less than the maximum depth of the vertices in T, then G_n' contains T as a subtree with the same root. Hence the claim follows. Similarly, since $G_n'[n]$ is a partial C-tree of depth n, $\lim_{n\to\infty} G_n'[n]$ coincides with a full C-tree. Since G_C is also a full C-tree and thus is isomorphic to both $\lim_{n\to\infty} G_n'$ and $\lim_{n\to\infty} G_n'[n]$, it follows from (4.5) and the definitions of $\lim_{n\to\infty} G_n$ and $\lim_{n\to\infty} \text{MARK}(G_n[n], X)$ that

$$\begin{aligned}&\lim_{n\to\infty} \text{MARK}(G_n(V_n, A_n)[n], X) \\ &\quad \preccurlyeq \lim_{n\to\infty} G_n(V_n, A_n) \preccurlyeq \text{MARK}(G_C(V_C, A_C), X).\end{aligned} \tag{4.6}$$

Since $G_n[n]$ is a partial C-tree of depth n, $G_{n+1}[n+1]$ contains $G_n[n]$ as a subtree with the same root. Thus it holds that $\text{MARK}(G_n[n], X) \preccurlyeq \text{MARK}(G_{n+1}[n+1], X)$; that is, the infinite sequence $\text{MARK}(G_N[N], X)$, $\text{MARK}(G_{N+1}[N+1], X)$, ... is *monotonic* with respect to $\preccurlyeq$. Hence it must hold that

$$\begin{aligned}&\lim_{n\to\infty} \text{MARK}(G_n(V_n, A_n)[n], X) \\ &\quad \equiv \text{MARK}(\lim_{n\to\infty} G_n(V_n, A_n)[n], X).\end{aligned}$$

Since $\lim_{n\to\infty} G_n(V_n, A_n)[n]$ is isomorphic to G_C, it also holds that

$$\begin{aligned}&\text{MARK}(\lim_{n\to\infty} G_n(V_n, A_n)[n], X) \\ &\quad \equiv \text{MARK}(G_C(V_C, A_C), X).\end{aligned}$$

By these two equations and (4.6), it follows that

$$\lim_{n\to\infty} G_n(V_n, A_n) \equiv \text{MARK}(G_C(C_C, A_C), X). \tag{4.7}$$

Since each replacement in Step 3.2 occurs at a deeper location than any subtree within depth l_1' of the root, the marked status of any vertex of depth less than or equal to l_1' remains unchanged throughout the execution of the body of EXTEND; that is,

$$\begin{aligned}&G_N(V_N, A_N)[l_1'] \\ &\quad \equiv \text{MARK}(G_C(V_C, A_C)[l_1' + l_2 + L_2], X)[l_1']\end{aligned} \tag{4.8}$$

for any N not less than $l_1' + l_2 + L_2$. Hence, Lemma 11 follows from (4.7) and (4.8), and we have the following.

THEOREM 3. *Let $F \cup \{C(X \rightarrow Y)\}$ denote a finite set of PFDs over a given class schema* S. *Then it is decidable whether or not $F \models C(X \rightarrow Y)$.*

5. ON COMPUTING CLOSURES

Let X denote a finite subset of PathFuncs(C). Although the closure X^+ may be an infinite subset of PathFuncs(C), the decidability proof in the previous section suggests a means of characterizing X^+. In fact, in this section, we derive an effective procedure for constructing a finite automaton which aspects X^+, and therefore prove that X^+ forms a regular set.

To begin, let $\tilde{G}_C = \text{MARK}(G_C, X)$, where G_C is a C-tree. We can view $\tilde{G}_C$ as a (possibly infinite) *automaton* by letting Root($\tilde{G}_C$) be the initial state, each marked vertex an accepting state, and each unmarked vertex a non-accepting state. Then the automaton accepts X^+ by Lemma 10 (with the simple convention that the automaton ignores any "dots" which occur in argument path functions). Clearly, this automaton is finite iff $\tilde{G}_C$ is finite. For the remainder of this section, we focus on the more difficult case that arises when $\tilde{G}_C$ is infinite. As a matter of convenience, we shall assume that the reader is reasonably familiar with any other notation introduced in the previous section.

Our overall strategy will be to modify $\tilde{G}_C$ to a finite automaton by redirecting various arcs. An informal example should help to clarify the main ideas behind this strategy. To begin, let S consist of the complex object types

```
a{B:b, C:c}
b{A:a}
c{ }
```

and consider where F consists of the single PFD,

```
a(C → B.A.C),
```

and where $X = \{C\}$. An a-tree G_a (which is infinite) is illustrated in Fig. 12. Now consider $\tilde{G}_a(\tilde{V}_a, \tilde{A}_a) = \text{MARK}(G_a, X)$. It is straightforward to confirm that ver-

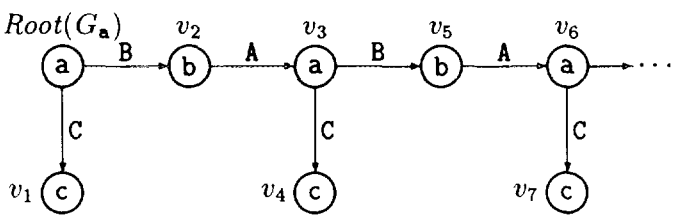

FIG. 12. An a-tree G_a.

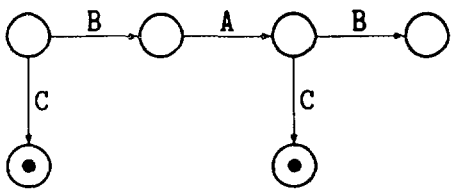

FIG. 13. Marking pattern of a subtree of depth at least 3 with root labeled "a."

tices in $\tilde{V}_a$ labeled "c" (e.g., v_1, v_4, and v_7) will be marked and all others unmarked. Consider each subtree of $\tilde{G}_a$ of depth at least 3 ($= l_2$) with a root vertex labeled "a." Each such subtree will have the marking pattern illustrated in Fig. 13 in which the marked vertices correspond to the vertices with bullets in the pattern. Thus, each (infinite) subtree whose root is a vertex labeled "a" has the same marking pattern. (This will be formally proven below.) We can therefore represent the marking pattern of $\tilde{G}_a$ by *redirecting* the destination of the out-arc of vertex v_5 from v_6 to v_3. For this graph, we can construct a finite automaton accepting C^+ as follows: (1) let Root($\tilde{G}_a$) be the initial state, (2) let the two marked vertices v_1 and v_4 be accepting states, and (3) let all unmarked vertices Root($\tilde{G}_a$), v_2, v_3, and v_5 be non-accepting states.[5] The resulting automaton is illustrated in Fig. 14. Clearly, given $pf \in \text{PathFuncs}(a)$, the automaton can decide in $O(\text{len}(pf))$ time whether or not $pf \in C^+$.

LEMMA 14. *Let v_1 and v_2 be vertices in $\tilde{V}_C$ such that* (1) $l_{C1}(v_1) = l_{C1}(v_2)$, (2) *the depths of v_1 and v_2 are greater than l_1, and* (3) $T_C(v_1)[l_2] \equiv T_C(v_2)[l_2]$. *Then* $T_C(v_1) \equiv T_C(v_2)$.

Proof. By symmetry and assumption, it suffices to show that

$$T_C(v_1) - T_C(v_1)[l_2] \preccurlyeq T_C(v_2) - T_C(v_2)[l_2]. \quad (5.1)$$

A proof of this is analogous to our proof of (4.4) in Lemma 13 and is left to the Appendix. ∎

In general, given $pf \in \text{PathFuncs}(C)$, we need to decide whether or not Root($\tilde{G}_C$)$\cdot pf$ is marked; that is, if Root($\tilde{G}_C$)$\cdot pf$ will qualify as an accepting state in the eventual automaton. Choose l_1 as the integer l_1', and assume that $\text{len}(pf) \geq l_1 + l_2 + L_2$. Then there are two distinct ancestors v_1, v_2 of Root($\tilde{G}_C$)$\cdot pf$ satisfying the three conditions of Step 3.1 in Procedure EXTEND, and it follows from Lemma 14 that $T_C(v_1) \equiv T_C(v_2)$. Thus, for every $pf' \in \text{PathFuncs}(l_{C1}(v_1))$, $v_1 \cdot pf'$ is marked iff $v_2 \cdot pf'$ is marked. Now consider that there must exist path functions pf_1, pf_2, and pf_3 such that (1) Root($\tilde{G}_C$)$\cdot pf_1 = v_1$, (2) Root($\tilde{G}_C$)$\cdot pf_1 \circ pf_2 = v_2$, (3) $pf = pf_1 \circ pf_2 \circ pf_3$, and (4)

[5] A simpler automaton exists in this case. In general, it is not part of our intention in this section to derive automatons with the fewest states.

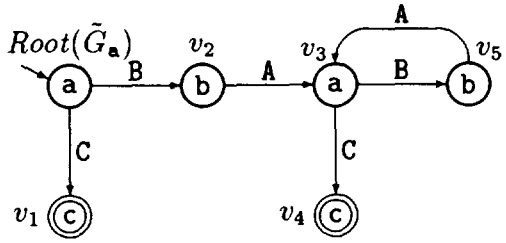

FIG. 14. A finite automaton accepting C^+.

Root($\tilde{G}_C$)$\cdot pf$ is marked iff $v_1 \cdot pf_3$ is marked (since $pf_3 \in$ PathFuncs($l_{Cl}(v_1)$)). With this observation in mind, consider the following procedure that navigates within $\tilde{G}_C(\tilde{V}_C, \tilde{A}_C)$ by following arcs labeled by properties in the sequence they occur in an argument path function.

PROCEDURE TRAVERSE($P_1 \cdot P_2 \cdot \cdots \cdot P_n$).

Input. A path function $P_1 \cdot P_2 \cdot \cdots \cdot P_n \in$ PathFuncs(C).

Output. A vertex $v \in \tilde{V}_C$ with the same marked status as vertex Root($\tilde{G}_C$)$\cdot P_1 \cdot P_2 \cdot \cdots \cdot P_n$.

Step 1. Let $v \leftarrow$ Root($\tilde{G}_C$).
Step 2. **for** $i \leftarrow 1$ **to** n **do**
begin
Step 2.1. Let $v \leftarrow v \cdot P_i$.
Step 2.2. **If** there is a proper ancestor u of v such that (1) $l_{Cl}(u) = l_{Cl}(v)$, (2) the depth of u is between $l_1 + 1$ and $l_1 + L_2$, and (3) $T_1(u)[l_2] \equiv T_1(v)[l_2]$ **then** let $v \leftarrow u$.
end

The important conditions satisfied by vertex v returned by this procedure are given by the following lemma.

LEMMA 15. (a) *The depth of v does not exceed $l_1 + L_2$ during a call to* TRAVERSE.

(b) *Assume that the three preconditions of Step* 2.2 *are satisfied when the value of v is v' during a call to* TRAVERSE. *Then v' is the shallowest vertex on the path from* Root($\tilde{G}_C$) *to v' that satisfies the three preconditions. That is, for any proper ancestor v'' of v', there is no proper ancestor u'' of v'' such that* (1) $l_{Cl}(u'') = l_{Cl}(v'')$, (2) *the depth of u'' is between $l_1 + 1$ and $l_1 + L_2$, and* (3) $T_1(u'')[l_2] \equiv T_1(v'')[l_2]$.

Proof. Part (b) of the lemma is a straightforward consequence of the fact that v is reassigned to an ancestor vertex as soon as the three preconditions of Step 2.2 are satisfied. With regard to part (a) of the lemma, assume conversely that the depth of the vertex referenced by v, say v', exceeds $l_1 + L_2$. By virtue of the value L_2, there must then exist two distinct proper ancestors v_1 and v_2 of v' satisfying the three conditions of Step 3.1 of Procedure EXTEND. Since v is either reassigned to a child vertex in Step 2.1 or to an ancestor vertex in Step 2.2, v must necessarily have "visited" every ancestor of v'. Thus, since v_2 is an ancestor for which the three preconditions of Step 2.2 are satisfied, TRAVERSE will never visit any proper descendant of v_2, including v', which contradicts our assumptions. ∎

Now choose $l_1 + l_2 + L_2$ as the integer l_1', and let $\tilde{G}(\tilde{V}, \tilde{A})$ be computed as

$$\text{MARK}(G_C[l_1' + l_2 + L_2], X)[l_1 + l_2 + L_2].$$

Then, by Lemma 11,

$$\tilde{G} \equiv \text{MARK}(G_C, X)[l_1 + l_2 + L_2]$$

and it follows from Lemma 15(a) that any call to another verion of TRAVERSE, navigating $\tilde{G}$, will return a vertex v which has the same marked status as Root($\tilde{G}_C$)$\cdot pf$. (Note that, although any vertex referenced by v in the body of the procedure is always in $\tilde{G}[l_1 + L_2]$ by Lemma 15(a), the additional vertices in $\tilde{G} - \tilde{G}[l_1 + L_2]$ are still required in order to ensure that the third precondition of Step 2.2 remains effective.) Thus, since $G_C[l_1' + l_2 + L_2]$ is finite, $\tilde{G}$ can easily be constructed, and we can then use this new version of TRAVERSE as the means of deciding the marked status of any vertex in $\tilde{V}_C$.

Let us now consider how to construct a finite automaton accepting X^+ from $\tilde{G}(\tilde{V}, \tilde{A})$. By Lemma 15(b), we can compute the set of ordered pairs (v', u') of vertices in $\tilde{V}$ such that, whenever the vertex referenced by v in the body of TRAVERSE becomes v' in Step 2.1, then it is changed into u' in Step 2.2. In fact, a pair (v', u') is in the set, say *Redirect*, iff it satisfies the following two conditions.

1. v' is the shallowest vertex on the path from Root($\tilde{G}$) to v' such that the three preconditions of Step 2.2 are satisfied.

2. u' is the proper ancestor of v' for which the three preconditions are satisfied.

Let (v', u') be in *Redirect*, and let w denote the parent vertex of v'. Then there is a unique arc $w \xrightarrow{P} v' \in \tilde{A}$ for some property P (since $\tilde{G}$ is a tree). Consider when the vertex referenced by v in the body of TRAVERSE is changed from w to v' in Step 2.1 by virtue of the assignment "$v \leftarrow w \cdot P$." By definition of the pair (v', u'), the vertex referenced by v will then subsequently be changed to u' in Step 2.2. The same effect can therefore be achieved by redirecting the destination of the arc $w \xrightarrow{P} v'$ from v' to u' and then to perform the assignment "$v \leftarrow w \cdot P$."

By these observations, a finite automaton accepting X^+ can therefore be effectively constructed from $\tilde{G}(\tilde{V}, \tilde{A})$ as follows:

1. Let Root($\tilde{G}$) be the initial state, let each marked vertex be an accepting state, and let each unmarked vertex be a non-accepting state.

2. For each pair(v', u') in *Redirect*, redirect the destination of arc $w \xrightarrow{P} v'$ from v' to u', where $w \xrightarrow{P} v' \in \tilde{A}$.

Hence, we have the following theorem and corollary.

THEOREM 4. *Let X denote a finite subset of* PathFuncs(C), *where* $C \in$ Classes(**S**) *for class schema* **S**. *Then there is an effective procedure for constructing a finite automaton that accepts* X^+.

COROLLARY 1. *The closure* X^+ *is regular.*

Note that the constructed finite automaton is *essentially deterministic* in the sense that there is neither an arc labeled `Id`, corresponding to a silent transition, nor a vertex which has two or more out-arcs with the same label. Thus, once the finite automaton accepting X^+ is generated, it can be decided in $O(\|Y\|)$ time, whether or not $F \models C(X \rightarrow Y)$, where $\|Y\|$ is the size of Y.

6. POLYNOMIAL TIME ALGORITHMS FOR IMPLICATION PROBLEMS

The decision procedure given in Section 4 is not efficient. In fact, it takes more than exponential time on the total size of **S**, F, and $C(X \rightarrow Y)$ in order to decide whether or not $F \models C(X \rightarrow Y)$. In this section, we will present two special cases which have polynomial time algorithms for deciding whether or not $F \models C(X \rightarrow Y)$.

To simplify matters in the remainder of this section, we first consider the problem of deciding membership of (arbitrary) path functions in PathFuncs(C), for some $C \in$ Classes(**S**). This can be accomplished by a simple transformation of a schema graph $G_S(V_S, A_S)$ for **S** into a finite automaton FA_C which accepts PathFuncs(C) in the sense outlined in the previous section. The transformation proceeds as follows. First, assign the vertex $v \in V_S$ such that $l_{C1}(v) = C$ as the initial state. (Recall that v must be unique by condition SG1 of Lemma 4.) And second, assign all vertices in V_S as accepting states. It then follows from Condition SG2 of Lemma 4 that FA_C is essentially deterministic, and that, for any (not necessarily well-formed) path function pf, $v \cdot pf \in V_S$ iff $pf \in$ PathFuncs(C). Hence, presuming that missing transitions will in fact go to an additional non-accepting state, FA_C decides in $O(\text{len}(pf))$ time whether or not $pf \in$ PathFuncs(C).

For example, let **S** consist of the following complex object types:

```
a{B:b, C:c, F:f}
b{A:a}
c{D:d, E:e}
d{ }
e{ }
f{ }
```
(6.1)

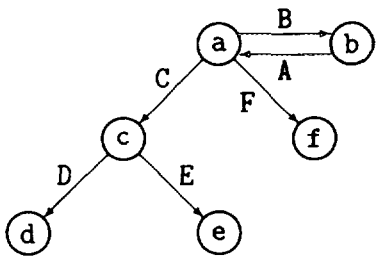

FIG. 15. A schema graph for S.

Then a schema graph for **S** will have the form illustrated in Fig. 15. The graph is transformed into the automaton FA_a by assigning the vertex labeled "a" as the initial state and by assigning all vertices as accepting states. Then FA_a decides in $O(\text{len}(pf))$ time whether or not $pf \in$ PathFuncs(a). For example, it accepts the path function `B.A.C` which is in PathFuncs(a), but rejects the path function `C.D.E.F` which is not in PathFuncs(a) (since, as presumed, an "E" transition from the state labeled "d" goes to a non-accepting state).

We now return to the issue of efficient algorithms for the implication problem. Let $\tilde{G}_C = \text{MARK}(G_C, X)$, where G_C is a C-tree and where X is a finite subset of PathFuncs(C). $\tilde{G}_C$ was considered as an automaton accepting X^+ in Section 5. If $pf \in$ PathFuncs(C), then, in view of condition M2, it should be clear that no non-accepting state is entered after reaching an accepting state. Hence, a simple expedient is to presume for any such automaton that, once an accepting state is entered, the remaining input is skipped and the automaton terminates with an "accept" status. We shall refer to such a machine as an *acceptor* of X^+ in the following discussion. Of course, the acceptor will only work properly if the input is in PathFuncs(C), but this can be resolved efficiently with the use of the automaton FA_C.

A slight variation of Procedure GENMARK given earlier yields an acceptor of X^+.

PROCEDURE ACCEPTOR(X).

Input. A finite subset X of PathFuncs(C).

Output. An acceptor of X^+.

Step 1. (Initialization). Assign a C-tree $G_C(V_C, A_C)$ to $G(V, A)$, let Root(G) be the initial state, and let all vertices in V be non-accepting states.

Step 2. For each $pf \in X$ such that Root(G) $\cdot$ pf is in V, change Root(G) $\cdot$ pf into an accepting state, and remove all the proper descendants and their incident arcs.

Step 3. Apply the following rule to G exhaustively:

AC. If v is a vertex in V and $C'(Z \rightarrow pf)$ a PFD in F such that (1) $C' = l_{C1}(v)$, (2) $v \cdot pf$ is in V and is a non-accpecting state, and (3) each path function in Z has a

form $pf_1 \circ pf_2$ such that $v \cdot pf_1$ is an accepting state,[6] then change $v \cdot pf$ into an accepting state and remove all proper descendants and their incident arcs.

The correspondence with Procedure GENMARK is as follows: rule M1 corresponds to Step 2 of this new procedure, while rules M2 and M3 correspond to rule AC in Step 3. Note that the possibility of factoring all applications of rule M1 prior to applications of rules M2 or M3 follows simply from the observation that we may exchange any application of rule M1 with a preceding application of either rule M2 or rule M3 without affecting the resulting computation of MARK(G, X) (since the preconditions for rule M1 are independent of the marked status of vertices in V). Thus, it can be proven with the same line of argument used in the proof of Lemma 10 that the procedure yields an acceptor of X^+. Also in common with GENMARK, the definition of ACCPETOR above is clearly not constructive for cases in which the C-tree G_C is infinite.

In the remainder of this section, we consider two cases in which the acceptor of X^+ can be constructed efficiently. The first case occurs if all antecendent PFDs (i.e., members of F) are key PFDs. Also, we shall continue to presume, without loss of generality, that the right-hand side of any member of F consists of a single path function.

Consider an application of rule AC to G in Step 3. In particular, assume that there exists a vertex $v \in V$ and a PFD $C'(Z \to pf) \in F$ such that (1) $C' = l_{C_1}(v)$, (2) $v \cdot pf$ is in V and is a non-accepting state, and (3) each path function in Z has a form $pf_1 \circ pf_2$ such that $v \cdot pf_1$ is an accepting state. Since $C'(Z \to pf)$ is a key PFD, there exists a path function pf' such that $pf \circ pf' \in Z$. Conditions (2) and (3) then imply that there exist path functions pf_3 and pf_4, where $pf' = pf_3 \circ pf_4$ and where $v \cdot pf \circ pf_3$ is an accepting state which is a proper descendant of $v \cdot pf$. Rule AC then implies an update to the automaton in which $v \cdot pf$ is changed to an accepting state and in which all proper descendants of $v \cdot pf$ (including $v \cdot pf \circ pf_3$) are removed from G. Now let

$$\text{Prefix}(X) \stackrel{\text{def}}{=} \{\text{Root}(G) \cdot pf' \mid \text{there exist } pf'' \text{ such that } pf' \circ pf'' \in X\}$$

Then Prefix(X) is a finite subset of V. At the end of Step 2 of ACCEPTOR, the initial set of accepting states will be a subset of $\{\text{Root}(G) \cdot pf \mid pf \in X\}$ (which in turn is a subset of Prefix(X)). By the observation above, the set of accepting states must continue to be a subset of Prefix(X) during execution of Step 3. Hence, the set of accepting states in ACCEPTOR(X), denoted S_{accept} in the following, is also a subset of Prefix(X). In order to determine if rule AC implies that a non-accepting state should be changed to an accepting state, one need only record the set of present accepting states. That is, in order to compute S_{accept}, it suffices to keep at most Prefix(X).

With this in mind, consider the time complexity for computing S_{accept}. Clearly, during initialization, there is no need to construct an entire C-tree G_C. Only a subtree induced by Prefix(X) needs to be created. Such a subtree is a partial C-tree $G(V, A)$ with $V = \text{Prefix}(X)$ and $A = \{u \xrightarrow{P} v \in A_C \mid u, v \in \text{Prefix}(X)\}$. Since the size of Prefix(X) is $\|X\|$, we can construct in $O(\|X\|)$ time a partial C-tree induced by Prefix(X). Thus, initialization requires $O(\|X\|)$ time. Since the partial C-tree is essentially deterministic, Step 2 also requires $O(\|X\|)$ time. Furthermore, for a given vertex v and PFD $C'(Z \to pf)$, it can be decided in $O(\|Z \cup \{pf\}\|)$ time whether or not the PFD satisfies the three preconditions of rule AC with respect to v. Hence, deciding whether or not rule AC applies for a given vertex requires $O(\|F\|)$ time. Since (1) the preconditions of rule AC cannot be satisfied by any leaf vertex, and (2) the number of internal vertices in the tree is at most $\|X\| - |X| + 1$, where $|X|$ is the cardinality of X, one application of rule AC requires $O(\|F\|(\|X\| - |X| - 1))$ time. Also, the number of internal vertices decreases each time this rule is applied to a vertex, which implies a bound of $\|X\| - |X| + 1$ on the number of times rule AC can apply. Thus, Step 3 requires $O(\|F\|(\|X\| - |X| + 1)^2)$ time. Hence, S_{accept} can also be computed in that time. Note that S_{accept} is sufficient for constructing an acceptor of X^+, even if the number of non-accepting states is infinite. Consequently, we have the following theorem.

THEOREM 5. *If F consists entirely of key PFDs, then an acceptor of X^+ can be constructed in $O(\|F\|(\|X\| - |X| + 1)^2)$ time.*

For example, let S consist of the six complex object types (6.1) above, and let F consist of the following three key PFDS:

$$f_1\text{: a(B C.E} \to \text{C)}$$
$$f_2\text{: a(B.A.C.E C.D} \to \text{B.A)}$$
$$f_3\text{: b(A} \to \text{Id)}.$$

We construct an acceptor of X^+ for the subset X of PathFuncs(a), consisting of the following path functions:

$$\text{B.A.C.E, B.A.C, B.A.F, C.D, C.E} \qquad (6.2)$$

A partial a-tree induced by Prefix(X) is illustrated in Fig. 16. After Step 1 of Procedure ACCEPTOR, vertex Root(G) is assigned as the initial state and all vertices as non-accepting states. In Step 2, five vertices v_4 to v_8 are changed into accepting states. For example, v_8 is changed into an accepting state because of the path function

[6] If $Z = \varnothing$, then condition (3) holds trivially.

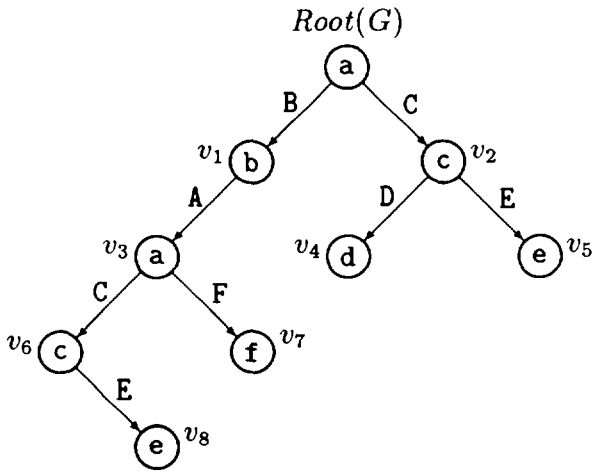

FIG. 16. A tree induced by Prefix(X).

B.A.C.E in X. However, the process of changing v_6 into an accepting state (because of B.A.C) will have the side-effect of removing vertex v_8.

In Step 3, the preconditions for an application of rule AC are now satisfied by vertex Root(G) and PFD f_2, since (1) l_{Cl}(Root(G)) = a, (2) Root(G).B.A ($=v_3$) is a non-accepting state for the right-hand side B.A of f_2, and (3) Root(G).B.A.C ($=v_6$) and Root(G).C.D ($=v_4$) are accepting states for the left-hand side B.A.C.E, C.D of f_2. Applying this rule to Root(G) and f_2 will then have the effect of changing vertex v_3 into an accepting state and of removing vertices v_6 and v_7. Rule AC can then be applied a second time to vertex v_1 and PFD f_3. Following this application, v_1 is itself changed into an accepting state, and vertex v_3 is removed. The third and final application of this rule concerns vertex Root(G) and PFD f_1. This will have the effect of changing vertex v_2 into an accepting state and of removing vertices v_4 and v_5.

The resulting acceptor of X^+ appears in Fig. 17. Note that Root(G) is the initial (non-accepting) state and that the remaining vertices, v_1 and v_2, are accepting states.

A second case in which the acceptor of X^+ can be constructed efficiently relates to the more specific circumstance in which F consists of key PFDs which are also simple, that is, in which each key PFD in F has the single identity path

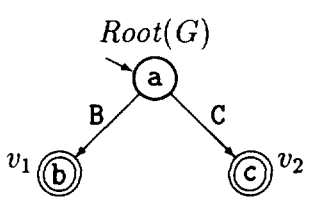

FIG. 17. An acceptor of X^+.

function Id occuring on its right-hand side. In this case, S_{accept} can be computed more efficiently as explained below.

Assume we are computing S_{accept} with the use of ACCEPTOR in the manner outlined above, and we let v be a non-accepting vertex for which no combination of a descendant of v (assuming that v also qualifies as a descendant) and key PFD in F satisfies the three preconditions of rule AC in ACCEPTOR. We claim that no subsequent application of this rule, during the remaining compution of S_{accept}, will assign v as an accepting state if the key PFDs in F are simple. To prove this claim, it suffices to show that no non-accepting descendant of v is changed into an accepting state by a subsequent application of rule AC. To see this, consider any such subsequent application applied to a vertex u and a key PFD in F. Then u itself is changed into an accepting state and its proper descendants are removed (since the right-hand side of the PFD is Id by assumption). Now, if u is an ancestor of v, then v will be one of the proper descendants of u which is removed, and the claim continues to hold in this case. Otherwise, if u is not an ancestor of v, then u is not a descendant of v by assumption, and, again, the claim continues to hold.

Now assume that the possible application of rule AC for vertices is considered in a *bottom-up* fashion. By the claim above, there will never be any need to "return" to any vertex v previously considered if, at an earlier time, it had been confirmed that no PFD in F, together with v, satisfied the preconditions of rule AC. Alternatively, if the preconditions of rule AC had been satisfied by v and some PFD in F, then v will have been made a leaf.

These observations imply that S_{accept} can be computed by considering the possible application of rule AC of ACCPETOR for each non-accepting vertex *at most once* in a bottom-up fashion. Hence, the time for rule AC is reduced to $O(\|F\|(\|X\| - |X| + 1))$. Therefore, S_{accept} can also be computed in that time, and the following theorem holds.

THEOREM 6. *If every PFD in F is a key PFD which is simple, then an acceptor of X^+ can be constructed in $O(\|F\|(\|X\| - |X| + 1))$ time.*

For example, reconsider the construction of X^+ given above in which S and X consist of the six complex object types (6.1) and five path functions (6.2), respectively, but now assume that F consists of the following three PFDs (note that each is a key PFD which is also simple):

f_4: a(B.A C.E → Id)

f_5: a(C.E F → Id)

f_6: b(A → Id).

The first two steps of ACCEPTOR will have the same effect for the partial a-tree in Fig. 16. As we have suggested for

rule AC, each vertex in the tree is then considered in a bottom-up fahsion to see if the vertex, together with any of f_4, f_5 or f_6, satisfies the preconditions of this rule. To begin, assume that vertex v_3 is the first vertex considered (vertex v_2 would also qualify). Clearly, v_3 and PFD f_5 satisfy the preconditions of rule AC, and therefore v_3 is assigned as an accepting state, and vertices v_6 and v_7 are removed. Note that no further consideration of v_3 is necessary since v_3 is now a leaf vertex.

Assume that vertex v_2 is the next vertex considered (vertex v_1 would now also qualify). In this case, however, neither f_4, f_5, nor f_6, together with v_2, satisfies the preconditions of rule AC, and therefore v_2 remains as a non-accepting state. Again note that no further consideration of v_2 will be necessary.

The only possible choice for the next vertex to be considered is now v_1. In this case, v_1 and PFD f_6 satisfy the preconditions of rule AC. v_1 is therefore assigned as an accepting state, and vertex v_3 is removed.

Finally, vertex Root(G) must be considered. In this case, Root(G) and PFD f_4 satisfy the preconditions of rule AC. This will cause Root(G) to be assigned as an accepting state and all other vertices to be removed. Thus, an acceptor of X^+ consists of a single vertex Root(G), which is the initial (accepting) state. Hence, X^+ coincides with PathFuncs(a).

Note that, with the earlier construction of X^+, it would not have been possible to consider vertices in a purely bottom-up fashion. In particular, vertex v_1 and PFD f_3 would not satisfy the preconditions of rule AC unless the rule is applied in advance to vertex Root(G) (a proper ancestor of v_1) and PFD f_2.

Note that an acceptor of X^+ constructed in a manner outlined in this section is essentially deterministic. Thus, once the acceptor X^+ is constructed, for a given PFD $C(X \rightarrow Y)$, it can be decided in $O(\| Y \|)$ time whether or not $F \models C(X \rightarrow Y)$.

7. SUMMARY AND OPEN PROBLEMS

In order to overcome several problems with the relational model when used for complex applications, semantic or object-oriented data models support the definition of complex object types with at least two properties. First, any object of a given type is assumed to have an identity separate from any of its parts; and second, the parts themselves may be the same or any other objects. The notion of a *path functional dependency* (or PFD) in which component attributes correspond to descriptions of property value paths in such object bases was first proposed and considered in [20]. The main contribution of this earlier work was a sound and complete axiomatization when databases may be infinite. In this paper, we have resolved a number of issues which were left open:

- We have proven that the same axiomatization remains complete when PFDs are permitted empty lef-hand sides. In our introductory comments, we reviewed an application of PFD theory which makes use of such constraints.
- We have shown that the axiomatization is not complete if logical consequence is defined with respect to finite databases only.
- We have resolved the issue of decidability of logical implication for PFDs in the affirmative. Our proof suggested that an important functional closure forms a regular set, which lead us to the derivation of an effective procedure for constructing a deterministic finite state automaton accepting the set.
- We have derived efficient polynomial time algorithms for the implication problem based on this procedure which apply in cases where antecedent PFDs are a form of complex or embedded *key* constraint.

Some issues that remain unresolved include the following:

- *Complexity of the general membership problem for PFDs.* Given a finite set $F \cup \{C(X \rightarrow Y)\}$ of PFDs over a class schema, is it NP-hard (or NP-complete) to decide whether or not $F \models C(X \rightarrow Y)$? The issue remains unresolved even if one restricts schema to be acyclic.
- *A finitely complete axiomatization.* Find a complete set of inference axioms for finite logical implication for PFDs.
- *Decidability and complexity issues for finite logical implication.* Given a finite set $F \cup \{C(X \rightarrow Y)\}$ of PFDs over a class schema, is it (efficiently) decidable whether or not $F \models_{\text{finite}} C(X \rightarrow Y)$? The issue remains unresolved even if $F \cup \{C(X \rightarrow Y)\}$ consists of only simple key PFDs.

In view of past experience on finite implication problems for the relational model, we expect that problems in the latter two categories will be very hard.

There is one final point worth noting about our underlying data model which relates to the concept of *generalization*. Semantic or object-oriented data models usually allow the definition of a class (or object type) to mention at least one *superclass* (or *supertype*)—more than one if the model supports so-called *multiple inheritance*. One of the authors has extended the earlier work on PFDs in [20] to a more general model in which complex object types can also be organized in an arbitrary generalization taxonomy [19]. In particular, this later work permitted a complex object type to include an additional "`isa`" clause. For example, a `grad` complex object type for the UNIVERSITY schema could be defined as

```
grad{Sup:prof} isa {student, prof}.
```

It is straightforward to extend the results of this paper to this more general model.

APPENDIX: PROOF OF (5.1) IN LEMMA 14

By condition CT1 of Lemma 3, for a vertex u in $T_C(v_1)$, there is a path function $pf \in \text{PathFuncs}(l_{C1}(v_1))$ such that $v_1 \cdot pf = u$. To simplify the notation, assume that u' denotes the corresponding vertex $v_2 \cdot pf$ in $T_C(v_2)$ and let u be a marked vertex in $T_C(v_1) - T_C(v_1)[l_2]$. Then (5.1) follows if u' must also be marked, which we prove by induction on the sequence of applications of rules M1 to M3 during the execution of Step 2 for an invocation of function GENMARK on a C-tree G_C and finit subset X of PathFuncs(C).

Basis. Since every vertex in V_C is initially unmarked in Step 1, (5.1) holds trivially.

Induction. Consider where vertex u must become marked by virtue of the nth application of one of the rules M1 to M3. By the induction hypothesis, we may assume that, for $m < n$, if the mth application of a rule in Step 2 changes vertex w in $T_C(v_1) - T_C(v_1)[l_2]$ to a marked status, then w' is also marked in $T_C(v_2) - T_C(v_2)[l_2]$. There are three cases to be considered.

Case 1. (Where rule M1 applies). Then u must coincide with $\text{Root}(\tilde{G}_C) \cdot pf$ for some $pf \in X$. Since (1) the depth of v_1 is greater than l_1 by assumption and (2) $\text{len}(pf) \leqslant l_1$, the depth of v_1 is greater than the depth of $\text{Root}(\tilde{G}_C) \cdot pf$. Thus $\text{Root}(\tilde{G}_C) \cdot pf$ is in $\tilde{G}_C(\tilde{V}_C, \tilde{A}_C) - T_C(v_1)$. However, this contradicts our assumption that u is in $T_C(v_1) - T_C(v_1)[l_2]$. Thus, this is not the case.

Case 2 (Where rule M2 applies). Then there is an ancestor w of u that has already been marked. There are three subcases to be considered.

Case 2.1 (Where w is in $T_C(v_1)[l_2]$). Then w' is in $T_C(v_2)[l_2]$, and thus marked by the assumption that $T_C(v_1)[l_2] \equiv T_C(v_2)[l_2]$. Since w' is an ancestor of u', by Claim 3 in the proof of Lemma 13, u' must be marked to satisfy condition M2.

Case 2.2 (Where w is in $T_C(v_1) - T_C(v_1)[l_2]$). Note that w has already been marked when u is changed to a marked status, and thus w' is marked by the induction hypothesis. Hence, as in Case 2.1, u' must be marked.

Case 2.3 (Where w is in $G_C(V_C, A_C) - T_C(v_1)$). Since $T_C(v_1)$ is a tree with root v_1 and w is an ancestor of u, the assumption implies that w is an ancestor of v_1. Thus, by Claim 3, v_1 as well as u should both be marked to satisfy condition M2 by virtue of vertex w. Furthermore, since v_1 is in $T_C(v_1)[l_2]$, v_2 is marked by the assumption that $T_C(v_1)[l_2] \equiv T_C(v_2)[l_2]$. Since v_2 is an ancestor of u', as in Case 2.1, vertex u' must be marked.

Case 3 (Where rule M3 applies). Then, for an ancestor w of u, there is a PFD $C'(Z \rightarrow pf) \in F$ such that $C' = l_{C1}(w)$, $w \cdot pf = u$, and $w \cdot pf_Z$ is marked for every $pf_Z \in Z$. Since (1) $\text{len}(pf) \leqslant l_2$ by choice of l_2 and (2) u is in $T_C(v_1) - T_C(v_1)[l_2]$ by assumption, the ancestor w of u must be in $T_C(v_1)$ in order that $w \cdot pf = u$. Thus, each marked vertex $w \cdot pf_Z$ is also in $T_C(v_1)$. It can be proven along the same line of argument for Cases 2.1 and 2.2 above that each corresponding vertex $w' \cdot pf_Z$ is marked. Hence, by Claim 3, $w' \cdot pf$ must be marked to satisfy condition M3 by virtue of vertex w' and the PFD. Then (5.1) follows since $u' = w' \cdot pf$.

ACKNOWLEDGMENT

The authors are grateful for the many excellent comments and suggestions by an anonymous referee; especially, the readability of Section 4 was significantly improved by the referee's suggestions.

REFERENCES

1. S. Abiteboul and S. Grumback, COL: A logic-based language for complex objects, *in* "Proceedings, 1st International Conference on Extending Database Technology (EDBT), March 1988," Lecture Notes in Computer Science, Vol. 303, pp. 271–293, Springer-Verlag, New York/Berlin, 1988.
2. S. Abiteboul and P. C. Kanellakis, Object identity as a query language primitive, *in* "Proceedings, ACM SIGMOD International Conference on Management of Data, June 1989," pp. 159–173.
3. A. V. Aho, Y. Sagiv, and J. D. Ullman, Efficient optimization of a class of relational expressions, *ACM Trans. Database Systems* **4**, No. 4 (1979), 435–454.
4. C. Beeri, Formal models for object-oriented databases, *in* "Proceedings, 1st International Conference on Deductive and Object-Oriented Databases (DOOD), December 1989," pp. 370–395.
5. C. Beeri, A formal approach to object-oriented databases, *Data Knowledge Engrg.* **5** (1990), 353–382.
6. C. Beeri and M. Y. Vardi, The implication problem for data dependencies, *in* "Proceedings, 8th International Colloquium on Automata, Languages and Programming (ICALP), July 1981," Lecture Notes in Computer Science, Vol. 115, pp. 73–85, Springer-Verlag, New York/Berlin, 1981.
7. M. F. van Bommel and G. E. Weddell, Reasoning about equations and functional dependencies on complex objects, *IEEE Trans. Knowledge Data Engrg.* **6**, No. 3 (1994), 455–469.
8. S. Cluet, C. Delobel, C. Lécluse, and P. Richard, RELOOP, an algebra based query language for an object-oriented database system, *in* "Proceedings, 1st International Conference on Deductive and Object-Oriented Databases (DOOD), December 1989," pp. 294–313.
9. U. Dayal, Queries and views in an object-oriented data model, *in* "Proceedings, 2nd International Workshop on Database Programming Languages, June 1989," pp. 80–102, Morgan Kaufmann, San Mateo, CA, 1989.

10. S. K. Debray and D. S. Warren, Functional computation in logic programs, *ACM Trans. Programming Lang. Syst.* **11**, No. 3 (1989), 451–481.
11. W. Kent, Limitations of record-based information models, *ACM Trans. Database Systems* **4**, No. 1 (1979), 107–131.
12. C. Lécluse, P. Richard, and Velez, O_2: An object-oriented data model, *in* "Proceedings, ACM SIGMOD International Conference on Management of Data, June 1988," pp. 424–433.
13. A. O. Mendelzon, Functional dependencies in logic programs, *in* "Proceedings, 11th International Conference on Very Large Data Bases (VLDB), August 1985," pp. 324–330.
14. A. O. Mendelzon and P. T. Wood, Functional dependencies in horn clause queries, *ACM Trans. Database Systems* **16**, No. 1 (1991), 31–55.
15. J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong, A language facility for designing database-intensive applications, *ACM Trans. Database Systems* **5**, No. 2 (1980), 185–207.
16. G. M. Shaw and S. B. Zdonik, An object-oriented query algebra, *in* "Proceedings, 2nd International Workshop on Database Programming Languages, June 1989," pp. 103–112, Morgan Kaufmann, San Mateo, CA, 1989.
17. D. W. Shipman, The functional data model and the data language daplex, *ACM Trans. Database Systems* **6**, No. 1 (1981), 140–173.
18. S. L. Vandenberg and D. J. DeWitt, Algebraic support for complex objects with arrays, *in* "Proceedings, ACM SIGMOD International Conference on Management of Data, June 1991," pp. 158–167.
19. G. E. Weddell, A theory of functional dependencies for object-oriented data models, *in* "Proceedings, 1st International Conference on Deductive and Object-Oriented Databases (DOOD), December 1989," pp. 150–169.
20. G. E. Weddell, Reasoning about functional dependencies generalized for semantic data models, *ACM Trans. Database Systems* **17**, No. 1 (1992), 32–64.

Printed in Belgium
Uitgever: Academic Press, Inc.
Verantwoordelijke uitgever voor België:
Hubert Van Maele
Altenastraat 20, B-8310 Sint-Kruis